


F2837xD IPC (Inter-Processor Communication) Device Driver

USER'S GUIDE



Copyright

Copyright © 2019 Texas Instruments Incorporated. All rights reserved. Other names and brands may be claimed as the property of others.

 Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this document.

Texas Instruments
13905 University Boulevard
Sugar Land, TX 77479
<http://www.ti.com/c2000>



Revision Information

This is version 3.06.00.00 of this document, last updated on Mon May 27 06:48:20 CDT 2019.

Table of Contents

Copyright	2
Revision Information	2
1 Introduction	5
2 Inter-processor Communication (IPC) Drivers	7
2.1 IPC API Drivers	7
2.2 IPC-Lite API Drivers	43
2.3 IPC Utility Drivers	55
IMPORTANT NOTICE	58

1 Introduction

The IPC (Inter-Processor Communication) Driver for Texas Instruments® F2837xD device provides a set of API functions for using the IPC module to communicate between the two C28 processors - from here on referred to as CPU1 and CPU2.

Functions and structures are provided for sending and receiving basic IPC commands via 2 solutions:

1. The main IPC drivers communicate through Put and Get ring buffers in the CPU1 to CPU2 and CPU2 to CPU1 MSG RAM's, and allow the user to queue up commands and use multiple interrupt service routines (ISR's) for IPC communications.
2. The IPC-Lite drivers communicate via the IPC registers only, and require no additional memory. They are limited to usage with a single IPC ISR, and commands cannot be queued one after another.

With very few exceptions, the same IPC APIs are used by both CPU1 and CPU2 processors. As a result all the APIs use the acronyms "LtoR" or "RtoL"

to represent "Local To Remote" and "Remote to Local" CPU access. For example if the function `IPCLtoRDataWrite` is called from CPU1, CPU1 would be the local whereas CPU2 would be the remote.

The API documentation for IPC drivers is located in C2000Ware under the `/device_support/f2837xd/docs/` directory.

The driver is contained in `/common/source/F2837xD_Ipc_Driver.c`, `/common/source/F2837xD_Ipc_Driver_Lite.c` and

`/common/source/F2837xD_Ipc_Driver_Util.c` with `/common/source/F2837xD_Ipc_Driver.h` containing the API definitions

for use by applications.

Note:

CPU2 TO CPU1 IPC INT0 and IPC Flag 31 are used by the CPU2 boot ROM to report system status errors to the CPU1 master system during CPU2 boot time. If the CPU1 application code uses CPU2 TO CPU1 IPC INT0, the application must either handle the CPU2 TO CPU1 Boot ROM status commands in the ISR (Recommended to ensure CPU2 has booted properly), or ignore the commands by setting the CPU2 TO CPU1 IPCACK bits for `IPC_FLAG0` and `IPC_FLAG31`.

For examples of how to use both the IPC and IPC-Lite drivers, see the examples in C2000Ware for your device.

2 Inter-processor Communication (IPC) Drivers

IPC API Drivers	7
IPC-Lite API Drivers	43
IPC Utility Drivers	55

2.1 IPC API Drivers

Data Structures

- [tlpcController](#)
- [tlpcMessage](#)

Functions

- `uint16_t` [IpcGet](#) (volatile [tlpcController](#) *psController, [tlpcMessage](#) *psMessage, `uint16_t` bBlock)
- `uint32_t` [IPCGetBootStatus](#) (void)
- void [IPCInitialize](#) (volatile [tlpcController](#) *psController, `uint16_t` usCPU2IpcInterrupt, `uint16_t` usCPU1IpcInterrupt)
- `uint16_t` [IPCLiteLtoRClearBits](#) (`uint32_t` ulFlag, `uint32_t` ulAddress, `uint32_t` ulMask, `uint16_t` usLength, `uint32_t` ulStatusFlag)
- `uint16_t` [IPCLiteLtoRClearBits_Protected](#) (`uint32_t` ulFlag, `uint32_t` ulAddress, `uint32_t` ulMask, `uint16_t` usLength, `uint32_t` ulStatusFlag)
- `uint16_t` [IPCLiteLtoRDataRead](#) (`uint32_t` ulFlag, `uint32_t` ulAddress, `uint16_t` usLength, `uint32_t` ulStatusFlag)
- `uint16_t` [IPCLiteLtoRDataWrite](#) (`uint32_t` ulFlag, `uint32_t` ulAddress, `uint32_t` ulData, `uint16_t` usLength, `uint32_t` ulStatusFlag)
- `uint16_t` [IPCLiteLtoRDataWrite_Protected](#) (`uint32_t` ulFlag, `uint32_t` ulAddress, `uint32_t` ulData, `uint16_t` usLength, `uint32_t` ulStatusFlag)
- `uint16_t` [IPCLiteLtoRFunctionCall](#) (`uint32_t` ulFlag, `uint32_t` ulAddress, `uint32_t` ulParam, `uint32_t` ulStatusFlag)
- `uint16_t` [IPCLiteLtoRGetResult](#) (void *pvData, `uint16_t` usLength, `uint32_t` ulStatusFlag)
- `uint16_t` [IPCLiteLtoRSetBits](#) (`uint32_t` ulFlag, `uint32_t` ulAddress, `uint32_t` ulMask, `uint16_t` usLength, `uint32_t` ulStatusFlag)
- `uint16_t` [IPCLiteLtoRSetBits_Protected](#) (`uint32_t` ulFlag, `uint32_t` ulAddress, `uint32_t` ulMask, `uint16_t` usLength, `uint32_t` ulStatusFlag)
- `uint16_t` [IPCLiteReqMemAccess](#) (`uint32_t` ulFlag, `uint32_t` ulMask, `uint16_t` ulMaster, `uint32_t` ulStatusFlag)
- void [IPCLiteRtoLClearBits](#) (`uint32_t` ulFlag, `uint32_t` ulStatusFlag)
- void [IPCLiteRtoLClearBits_Protected](#) (`uint32_t` ulFlag, `uint32_t` ulStatusFlag)
- void [IPCLiteRtoLDataRead](#) (`uint32_t` ulFlag, `uint32_t` ulStatusFlag)

- void [IPCLiteRtoLDataWrite](#) (uint32_t ulFlag, uint32_t ulStatusFlag)
- void [IPCLiteRtoLDataWrite_Protected](#) (uint32_t ulFlag, uint32_t ulStatusFlag)
- void [IPCLiteRtoLFunctionCall](#) (uint32_t ulFlag, uint32_t ulStatusFlag)
- void [IPCLiteRtoLSetBits](#) (uint32_t ulFlag, uint32_t ulStatusFlag)
- void [IPCLiteRtoLSetBits_Protected](#) (uint32_t ulFlag, uint32_t ulStatusFlag)
- uint16_t [IPCLtoRBlockRead](#) (volatile [tlpcController](#) *psController, uint32_t ulAddress, uint32_t ulShareAddress, uint16_t usLength, uint16_t bBlock, uint32_t ulResponseFlag)
- uint16_t [IPCLtoRBlockWrite](#) (volatile [tlpcController](#) *psController, uint32_t ulAddress, uint32_t ulShareAddress, uint16_t usLength, uint16_t usWordLength, uint16_t bBlock)
- uint16_t [IPCLtoRBlockWrite_Protected](#) (volatile [tlpcController](#) *psController, uint32_t ulAddress, uint32_t ulShareAddress, uint16_t usLength, uint16_t usWordLength, uint16_t bBlock)
- uint16_t [IPCLtoRClearBits](#) (volatile [tlpcController](#) *psController, uint32_t ulAddress, uint32_t ulMask, uint16_t usLength, uint16_t bBlock)
- uint16_t [IPCLtoRClearBits_Protected](#) (volatile [tlpcController](#) *psController, uint32_t ulAddress, uint32_t ulMask, uint16_t usLength, uint16_t bBlock)
- uint16_t [IPCLtoRDataRead](#) (volatile [tlpcController](#) *psController, uint32_t ulAddress, void *pvData, uint16_t usLength, uint16_t bBlock, uint32_t ulResponseFlag)
- uint16_t [IPCLtoRDataRead_Protected](#) (volatile [tlpcController](#) *psController, uint32_t ulAddress, void *pvData, uint16_t usLength, uint16_t bBlock, uint32_t ulResponseFlag)
- uint16_t [IPCLtoRDataWrite](#) (volatile [tlpcController](#) *psController, uint32_t ulAddress, uint32_t ulData, uint16_t usLength, uint16_t bBlock, uint32_t ulResponseFlag)
- uint16_t [IPCLtoRDataWrite_Protected](#) (volatile [tlpcController](#) *psController, uint32_t ulAddress, uint32_t ulData, uint16_t usLength, uint16_t bBlock, uint32_t ulResponseFlag)
- Uint16 [IPCLtoRFlagBusy](#) (uint32_t ulFlags)
- void [IPCLtoRFlagClear](#) (uint32_t ulFlags)
- void [IPCLtoRFlagSet](#) (uint32_t ulFlags)
- uint16_t [IPCLtoRFunctionCall](#) (volatile [tlpcController](#) *psController, uint32_t ulAddress, uint32_t ulParam, uint16_t bBlock)
- uint16_t [IPCLtoRSendMessage](#) (volatile [tlpcController](#) *psController, uint32_t ulCommand, uint32_t ulAddress, uint32_t ulDataW1, uint32_t ulDataW2, uint16_t bBlock)
- uint16_t [IPCLtoRSetBits](#) (volatile [tlpcController](#) *psController, uint32_t ulAddress, uint32_t ulMask, uint16_t usLength, uint16_t bBlock)
- uint16_t [IPCLtoRSetBits_Protected](#) (volatile [tlpcController](#) *psController, uint32_t ulAddress, uint32_t ulMask, uint16_t usLength, uint16_t bBlock)
- uint16_t [lpcPut](#) (volatile [tlpcController](#) *psController, [tlpcMessage](#) *psMessage, uint16_t bBlock)
- void [IPCRtoLBlockRead](#) ([tlpcMessage](#) *psMessage)
- void [IPCRtoLBlockWrite](#) ([tlpcMessage](#) *psMessage)
- void [IPCRtoLBlockWrite_Protected](#) ([tlpcMessage](#) *psMessage)
- void [IPCRtoLClearBits](#) ([tlpcMessage](#) *psMessage)
- void [IPCRtoLClearBits_Protected](#) ([tlpcMessage](#) *psMessage)
- void [IPCRtoLDataRead](#) (volatile [tlpcController](#) *psController, [tlpcMessage](#) *psMessage, uint16_t bBlock)
- void [IPCRtoLDataRead_Protected](#) (volatile [tlpcController](#) *psController, [tlpcMessage](#) *psMessage, uint16_t bBlock)
- void [IPCRtoLDataWrite](#) ([tlpcMessage](#) *psMessage)
- void [IPCRtoLDataWrite_Protected](#) ([tlpcMessage](#) *psMessage)
- void [IPCRtoLFlagAcknowledge](#) (uint32_t ulFlags)

- Uint16 [IPCRtoLFlagBusy](#) (uint32_t ulFlags)
- void [IPCRtoLFunctionCall](#) (tlpcMessage *psMessage)
- void [IPCRtoLSetBits](#) (tlpcMessage *psMessage)
- void [IPCRtoLSetBits_Protected](#) (tlpcMessage *psMessage)

2.1.1 Detailed Description

The main IPC drivers utilize circular buffers (1 Put buffer and 1 Get buffer for each IPC interrupt used) stored in the CPU1 to CPU2 and CPU2

to CPU1 message RAM's to send messages between processors. [tlpcController](#) data structures store the data for a single Put and Get buffer pair.

Note:

Although the main IPC drivers have the benefit of allowing queued commands for sequential processing, they require additional memory (message RAM's), code changes (command linker file changes and data structure variable definitions), and setup to begin using. For a simpler IPC driver solution which can be used instantly without any additional code changes, see the **IPC-Lite** API drivers.

The figure below shows how the IPC drivers are implemented in the dual core system. Note that buffers and indexes in purple store the CPU1 to CPU2 CPU1 Put buffer/ CPU2 Get

buffer and read/write indexes, while buffers and indexes in yellow store the CPU2 to CPU1 CPU2 Put buffer/ CPU1 Get buffer and read/write indexes . Additionally, each

CPU1 to CPU2 circular buffer is tied to a CPU1 to CPU2 IPC interrupt, and each CPU2 to CPU1 circular buffer is tied to a CPU2 to CPU1 IPC interrupt.

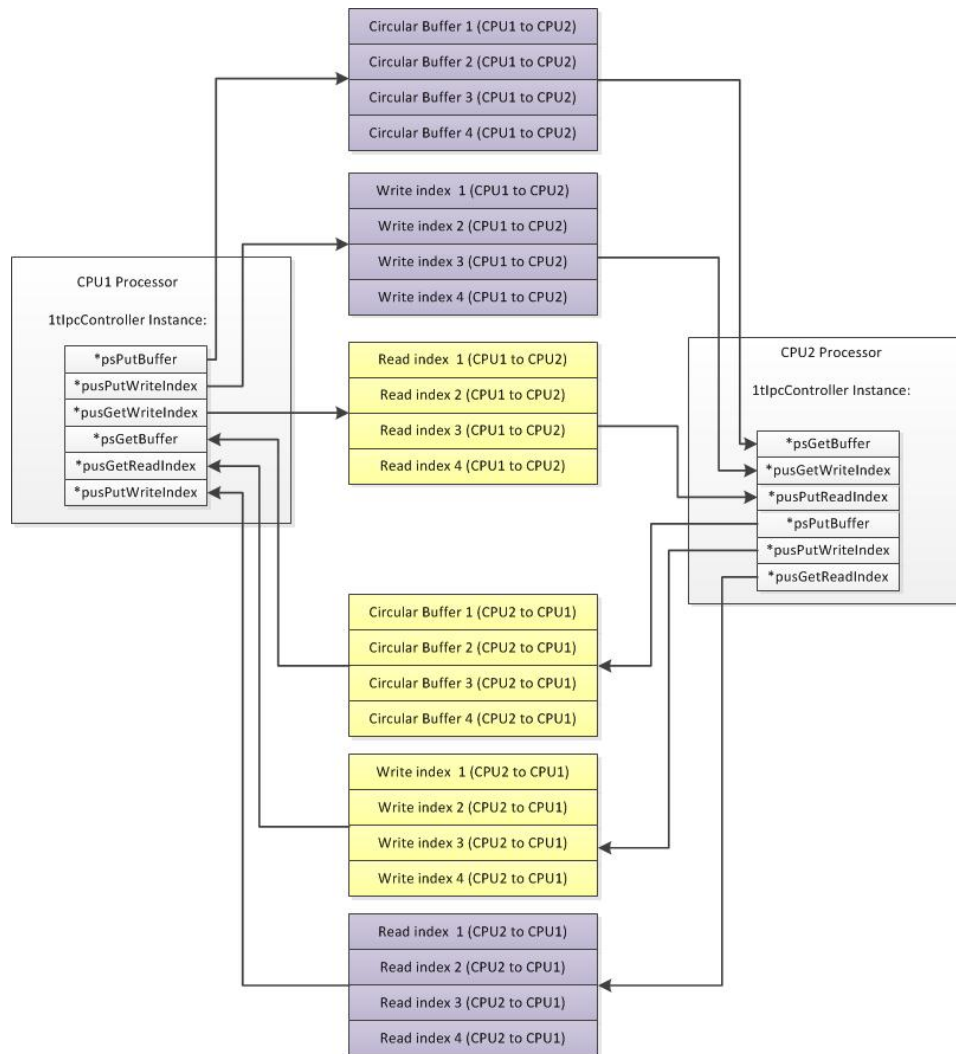


Figure 2.1: Main IPC Drivers Infrastructure Implementation

When one processor (X) wishes to perform an operation on the other processor (Y):

1. Processor X writes messages into its Put Buffer (equivalent to Processor Y's Get Buffer), increments the PutWriteIndex, and sets Processor Y's XtoY IPC interrupt flag.
2. Processor Y sees IPC interrupt flag and subsequently reads the messages from the Get Buffer and increments the GetReadIndex.
3. Processor Y then proceeds to process the commands according to the message contents and then acknowledges XtoY IPC interrupt flag.
4. Processor X can also read and process messages that Processor Y writes to its Put Buffer (which is Processor X's Get Buffer).

Messages are written to and read from each circular buffer in a FIFO fashion as shown below. N is the maximum number of messages that can be stored in a circular buffer and must be

an even number. By default, N= 4 in the main IPC drivers. There can be N-1 Puts into the circular buffer until the receiving processor's application code must perform a Get.

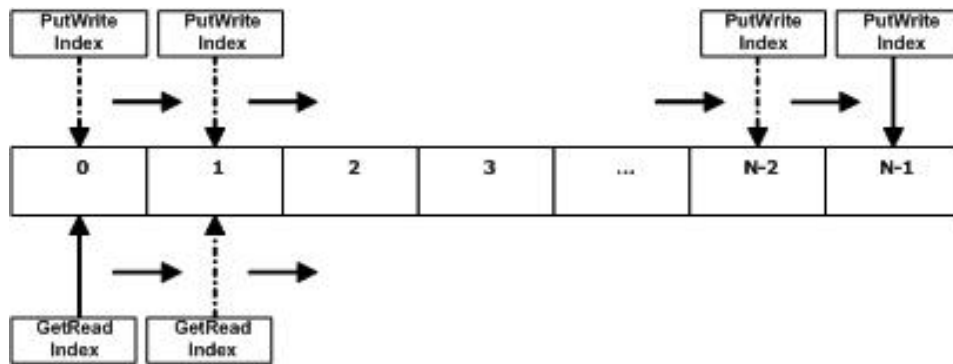


Figure 2.2: Writing to and Reading from Circular Buffer via Indexes

2.1.2 Usage and Examples

The IPC API driver functions will Put commands into the PutBuffer and Get commands from the GetBuffer in a manner that is transparent

to the application software. The application code is responsible for initializing the IPC controllers, calling the API driver functions

for those commands it wishes to execute, decoding received commands, and acknowledging received IPC interrupt flags.

There are a number of steps which must be taken in the application project in order to use the main IPC API driver functions.

1. Add the F2837xD_Ipc_Driver.c and F2837xD_Ipc_Driver_Util.c files which include the F2837xD_Ipc_drivers.h file to the application project. For C28 projects, make sure /common/source/F2837xD_Ipc_Driver.c and common/source/F2837xD_Ipc_Driver_Util.c are added as source files in your project, both of which include /common/include/F2837xD_Ipc_drivers.h. The F2837xD_Ipc_Driver_Util.c file includes utility functions that can be used by application code and that are used by both the main IPC driver functions and the IPC-lite driver functions.
2. Add IPC buffer and index section definitions to application project's .cmd command linker file. Memory regions for the CPU2 to CPU1 and CPU1 to CPU2 message RAM's should be defined. Additionally, there should be 1 section reserved for the PUTBUFFER, PUTWRITEIDX, and GETREADIDX, and 1 section reserved for the GETBUFFER, GETWRIDEIDX, PUTREADIDX which are mapped to the beginning of each message RAM block by grouping in the .cmd file.

The code below is a sample of what a CPU2 project's .CMD linker file should look like when using the main IPC API drivers:

```
MEMORY
{
    PAGE 0 :
        CPU2TOCPU1RAM      : origin = 0x03F800,   length = 0x000400
        CPU1TOCPU2RAM      : origin = 0x03FC00,   length = 0x000400
}
```

```
SECTIONS
{
  GROUP : > CPU2TOCPU1RAM,  PAGE = 1
  {
    PUTBUFFER
    PUTWRITEIDX
    GETREADIDX
  }

  GROUP : > CPU1TOCPU2RAM,  PAGE = 1
  {
    GETBUFFER :    TYPE = DSECT
    GETWRITEIDX :  TYPE = DSECT
    PUTREADIDX :   TYPE = DSECT
  }
}
```

The code below is a sample of what an CPU1 project's .CMD linker file should look like when using the main IPC API drivers:

```
MEMORY
{
  PAGE 0 :
    CPU2TOCPU1RAM : origin = 0x03F800, length = 0x000400
    CPU1TOCPU2RAM : origin = 0x03FC00, length = 0x000400
}

SECTIONS
{
  GROUP : > CPU1TOCPU2RAM,  PAGE = 1
  {
    PUTBUFFER
    PUTWRITEIDX
    GETREADIDX
  }

  GROUP : > CPU2TOCPU1RAM,  PAGE = 1
  {
    GETBUFFER :    TYPE = DSECT
    GETWRITEIDX :  TYPE = DSECT
    PUTREADIDX :   TYPE = DSECT
  }
}
```

1. Application source code must define and initialize at least 1 volatile global [tlpcController](#) instance on both the CPU1 and CPU2.

The IPC controller data structure ([tlpcController](#)) locally stores the pointers to the Put/Get buffers and their read/write indexes. Application code must define at least 1 instance of the [tlpcController](#) data structure on the CPU1 and a corresponding data structure on CPU2 for a single CPU1 IPC interrupt and CPU2 IPC interrupt communication pair (i.e. CPU1 will send

messages to CPU2 and inform CPU2 via a CPU2 IPC interrupt flag, and CPU2 will respond with a message sent to the CPU1 IPC interrupt flag). A code example of the definition and initialization of the [tlpcController](#) instance for CPU1 is shown below.

```
// *****
// At least 1 volatile global tIpcController instance is required
// when using IPC API Drivers.
// *****
volatile tIpcController g_sIpcController1;
volatile tIpcController g_sIpcController2;

void
main(void)
{
    // Initialize IPC Controllers
    IPCInitialize (&g_sIpcController1,  IPC_INT1,  IPC_INT1);
    IPCInitialize (&g_sIpcController2,  IPC_INT2,  IPC_INT2);
}
```

After these initialization steps, the CPU1 and CPU2 application code can set up an IPC interrupt service routine (ISR) corresponding to each

[tlpcController](#) instance that was defined, and then proceed to call the IPC command functions using the respective controllers. Generally

when an IPC command function is called for the sending processor, the responding function for the receiving processor has the same name. For

instance, the CPU1 application code might call the [IPCLtoRDataWrite\(\)](#) function to write data to memory accessible only by the CPU2. The CPU2 will

decode the command received in the Get Buffer message, and upon seeing an IPC_DATA_WRITE command, it will call [IPCRtoLDataWrite\(\)](#) function

to processes the command.

For IPC commands that must occur sequentially, the application code must use a single [tlpcController](#) instance when calling the sequential IPC

functions. This ensures that a single IPC interrupt processes the messages in the order they were received by the Get Buffer. If there is another

set of IPC commands that occurs independently from the first set of IPC commands, these can be called using a different [tlpcController](#) instance.

Note:

CPU2 TO CPU1 IPC INT0 and IPC Flag 31 are used by the CPU2 boot ROM to report system status errors to the CPU1 master system during CPU2 boot time.

If the CPU1 application code uses CPU2 TO CPU1 IPC INT0, the application must either handle the CPU2 TO CPU1 Boot ROM status commands in the ISR

(Recommended to ensure CPU2 has booted properly), or ignore the commands by setting the CPU2 TO CPU1 IPCACK bits for IPC_FLAG0 and IPC_FLAG31.

For examples demonstrating many of the basic IPC functions, see the IPC examples in the C2000Ware device_support package for your device.

2.1.3 Customization

There are a number of settings which can be customized for different application needs when using the main IPC API drivers in the header file

(F2837xD_lpc_drivers.h).

1. **IPC_BUFFER_SIZE:** This is the definition for the maximum number of [tlpcMessage](#) messages in the circular buffer. By default this value is 4, but it can be any even number as long as there is enough memory in the message RAM block for the buffer.
2. **NUM_IPC_INTERRUPTS:** This is the number of IPC interrupts for which an IPC driver API circular buffer is reserved (the number must be the same on both CPU1 and CPU2, because a single IPC controller needs information for 1 CPU1 IPC interrupt and 1 CPU2 IPC interrupt). This number is set to 4 by default. The value can be decreased to save message RAM memory if, for instance, one of the other interrupt channels is used for IPC-Lite function processing instead.
3. **tlpcMessage:** This is the data structure of the messages transmitted by the IPC API driver functions. Although the data structure is defined to expect a command, address, and two 32-bit data words for additional information as defined by the IPC API driver functions, the structure can also be used merely to transmit 4 generic 32-bit data words between processors, if desired. Application code will then have to process the Get accordingly in the IPC ISR.

2.1.4 Data Structure Documentation

2.1.4.1 tlpcController

Definition:

```
typedef struct
{
    tIpcMessage *psPutBuffer;
    uint32_t ulPutFlag;
    uint16_t *pusPutWriteIndex;
    uint16_t *pusPutReadIndex;
    tIpcMessage *psGetBuffer;
    uint16_t *pusGetWriteIndex;
    uint16_t *pusGetReadIndex;
}
tlpcController
```

Members:

psPutBuffer The address of the PutBuffer IPC message (in MSGRAM).
ulPutFlag The IPC INT flag to set when sending messages for this IPC controller instance.
pusPutWriteIndex The address of the PutBuffer Write index (in MSGRAM).
pusPutReadIndex The address of the PutBuffer Read index (in MSGRAM).
psGetBuffer The address of the GetBuffer IPC message(in MSGRAM).
pusGetWriteIndex The address of the GetBuffer Write Index (in MSGRAM).
pusGetReadIndex The address of the GetBuffer Read Index (in MSGRAM).

Description:

A structure that defines an IPC control instance. These fields are used by the IPC drivers, and normally it is not necessary for user software to directly read or write fields in the table.

2.1.4.2 tIpcMessage

Definition:

```
typedef struct
{
    uint32_t ulcommand;
    uint32_t uladdress;
    uint32_t uldataw1;
    uint32_t uldataw2;
}
tIpcMessage
```

Members:

ulcommand The command passed between processor systems.

uladdress The receiving processor address the command is requesting action on.

uldataw1 A 32-bit variable, the usage of which is determined by ulcommand. The most common usage is to pass length requirements with the upper 16-bits storing a Response Flag for read commands.

uldataw2 A 32-bit variable, the usage of which is determined by ulcommand. For block transfers, this variable is generally the address in shared memory used to pass data between processors.

Description:

A structure that defines an IPC message. These fields are used by the IPC drivers to determine handling of data passed between processors. Although they have a defined naming scheme, they can also be used generically to pass 32-bit data words between processors.

2.1.5 Function Documentation

2.1.5.1 IpcGet

Reads a message from the GetBuffer.

Prototype:

```
uint16_t
IpcGet(volatile tIpcController *psController,
       tIpcMessage *psMessage,
       uint16_t bBlock)
```

Parameters:

psController specifies the address of a *tIpcController* instance used to store information about the "Put" and "Get" circular buffers and their respective indexes.

psMessage specifies the address of the *tIpcMessage* instance where the message from GetBuffer should be written to.

bBlock specifies whether to allow function to block until GetBuffer has a message (1= wait until message available, 0 = exit with STATUS_FAIL if no message).

Description:

This function checks if there is a message in the GetBuffer. If so, it gets the message in the GetBuffer pointed to by the ReadIndex and writes it to the address pointed to by *psMessage*. The *bBlock* parameter can be one of the following values: **ENABLE_BLOCKING** or **DISABLE_BLOCKING**.

Returns:

STATUS_PASS if GetBuffer is empty. **STATUS_FAIL** if Get occurs successfully.

2.1.5.2 IPCGetBootStatus

Local Return CPU02 BOOT status

Prototype:

```
uint32_t  
IPCGetBootStatus(void)
```

Description:

This function returns the value at IPCBOOTSTS register.

Returns:

Boot status.

2.1.5.3 IPCInitialize

Initializes System IPC driver controller

Prototype:

```
void  
IPCInitialize(volatile tIpcController *psController,  
              uint16_t usCPU2IpcInterrupt,  
              uint16_t usCPU1IpcInterrupt)
```

Parameters:

psController specifies the address of a *tIpcController* instance used to store information about the "Put" and "Get" circular buffers and their respective indexes.

usCPU2IpcInterrupt specifies the CPU2 IPC interrupt number used by psController.

usCPU1IpcInterrupt specifies the CPU1 IPC interrupt number used by psController.

Description:

This function initializes the IPC driver controller with circular buffer and index addresses for an IPC interrupt pair. The *usCPU2IpcInterrupt* and *usCPU1IpcInterrupt* parameters can be one of the following values: **IPC_INT0**, **IPC_INT1**, **IPC_INT2**, **IPC_INT3**.

Note:

If an interrupt is currently in use by an *tIpcController* instance, that particular interrupt should not be tied to a second *tIpcController* instance.

For a particular *usCPU2IpcInterrupt* - *usCPU1IpcInterrupt* pair, there must be an instance of *tIpcController* defined and initialized on both the CPU1 and CPU2 systems.

Returns:

None.

2.1.5.4 IPCLiteLtoRClearBits

Sets the designated bits in a 16/32-bit data word at the remote CPU system address

Prototype:

```
uint16_t
IPCLiteLtoRClearBits(uint32_t ulFlag,
                     uint32_t ulAddress,
                     uint32_t ulMask,
                     uint16_t usLength,
                     uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Local to Remote IPC Flag number mask used to indicate a command is being sent.

ulAddress specifies the Remote CPU address to write to.

ulMask specifies the 16/32-bit mask for bits which should be set at the remote CPU ulAddress. (For 16-bit mask, only the lower 16-bits of ulMask are considered.)

usLength specifies the length of the **ulMask** (1 = 16-bit, 2 = 32-bit).

ulStatusFlag indicates the Local to Remote Flag number mask used to report the status of the command sent back from the Master system.

Description:

This function will allow the Local CPU system to set bits specified by the **usMask** variable in a 16/32-bit word on the Remote CPU system. The data word at /e ulAddress after the set bits command is then read into the IPCREMOTEREPLY register. After calling this function, a call to [IPCLiteLtoRGetResult\(\)](#) will read the data value in the IPCREMOTEREPLY register into a 16/32-bit variable in the Local CPU application. The **usLength** parameter accepts the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The **ulStatusFlag** parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**. The function returns **STATUS_PASS** if the command is successful or **STATUS_FAIL** if the request or status flags are unavailable.

Returns:

status of command (0=success, 1=error)

2.1.5.5 IPCLiteLtoRClearBits_Protected

Clears the designated bits in a 16/32-bit write-protected data word at Remote CPU system address

Prototype:

```
uint16_t
IPCLiteLtoRClearBits_Protected(uint32_t ulFlag,
                               uint32_t ulAddress,
                               uint32_t ulMask,
                               uint16_t usLength,
                               uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Local to Remote IPC Flag number mask used to indicate a command is being sent.

ulAddress specifies the Remote CPU write-protected address to write to.

ulMask specifies the 16/32-bit mask for bits which should be cleared at Remote CPU ulAddress. For 16-bit mask, only the lower 16-bits of ulMask are considered.

usLength specifies the length of the **ulMask** (1 = 16-bit, 2 = 32-bit).

ulStatusFlag indicates the Local to Remote Flag number mask used to report the status of the command sent back from the Master system.

Description:

This function will allow the Local CPU system to clear bits specified by the **usMask** variable in a write-protected 16/32-bit word on the Remote CPU system. The data word at /e ulAddress after the clear bits command is then read into the IPCREMOTEREPLY register. After calling this function, a call to [IPCLiteLtoRGetResult\(\)](#) will read the data value in the IPCREMOTEREPLY register into a 16/32-bit variable in the Local CPU application. The **usLength** parameter accepts the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The **ulStatusFlag** parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**. The function returns **STATUS_PASS** if the command is successful or **STATUS_FAIL** if the request or status flags are unavailable.

Returns:

status of command (0=success, 1=error)

2.1.5.6 IPCLiteLtoRDataRead

Reads either a 16- or 32-bit data word from the remote CPU System address

Prototype:

```
uint16_t
IPCLiteLtoRDataRead(uint32_t ulFlag,
                    uint32_t ulAddress,
                    uint16_t usLength,
                    uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Local to Remote IPC Flag number mask used to indicate a command is being sent.

ulAddress specifies the remote address to read from

usLength designates 16- or 32-bit read (1 = 16-bit, 2 = 32-bit)

ulStatusFlag indicates the Local to Remote Flag number mask used to report the status of the command sent back from the remote system.

Description:

This function will allow the Local CPU System to read 16/32-bit data from the Remote CPU System into the IPCREMOTEREPLY register. After calling this function, a call to [IPCLiteLtoRGetResult\(\)](#) will read the data value in the IPCREMOTEREPLY register into a 16- or 32-bit variable in the local CPU application. The **usLength** parameter accepts the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The **ulStatusFlag** parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**. The function returns **STATUS_PASS** if the command is successful or **STATUS_FAIL** if the request or status flags are unavailable.

Returns:

status of command (0=success, 1=error)

2.1.5.7 IPCLiteLtoRDataWrite

Writes a 16/32-bit data word to Remote CPU System address

Prototype:

```
uint16_t
IPCLiteLtoRDataWrite(uint32_t ulFlag,
                     uint32_t ulAddress,
                     uint32_t ulData,
                     uint16_t usLength,
                     uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Local to Remote IPC Flag number mask used to indicate a command is being sent.

ulAddress specifies the Remote CPU address to write to

ulData specifies the 16/32-bit word which will be written. For 16-bit words, only the lower 16-bits of ulData will be considered by the master system.

usLength is the length of the word to write (0 = 16-bits, 1 = 32-bits)

ulStatusFlag indicates the Local to Remote Flag number mask used to report the status of the command sent back from the Remote CPU system.

Description:

This function will allow the Local CPU System to write a 16/32-bit word via the *ulData* variable to an address on the Remote CPU System. The *usLength* parameter accepts the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The *ulStatusFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**. The function returns **STATUS_PASS** if the command is successful or **STATUS_FAIL** if the request or status flags are unavailable.

Returns:

status of command (0=success, 1=error)

2.1.5.8 IPCLiteLtoRDataWrite_Protected

Writes a 16/32-bit data word to a protected Remote CPU System address

Prototype:

```
uint16_t
IPCLiteLtoRDataWrite_Protected(uint32_t ulFlag,
                               uint32_t ulAddress,
                               uint32_t ulData,
                               uint16_t usLength,
                               uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Local to Remote IPC Flag number mask used to indicate a command is being sent.

ulAddress specifies the Remote CPU address to write to

ulData specifies the 16/32-bit word which will be written. For 16-bit words, only the lower 16-bits of ulData will be considered by the master system.

usLength is the length of the word to write (0 = 16-bits, 1 = 32-bits)

ulStatusFlag indicates the Local to Remote Flag number mask used to report the status of the command sent back from the Master system.

Description:

This function will allow the Local CPU System to write a 16/32-bit word via the *ulData* variable to a write-protected address on the Remote CPU System. The *usLength* parameter accepts the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The *ulStatusFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**. The function returns **STATUS_PASS** if the command is successful or **STATUS_FAIL** if the request or status flags are unavailable.

Returns:

status of command (0=success, 1=error)

2.1.5.9 IPCLiteLtoRFunctionCall

Calls a Remote CPU function with 1 optional parameter and an optional return value.

Prototype:

```
uint16_t
IPCLiteLtoRFunctionCall(uint32_t ulFlag,
                        uint32_t ulAddress,
                        uint32_t ulParam,
                        uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Local to Remote IPC Flag number mask used to indicate a command is being sent.

ulAddress specifies the Remote CPU function address

ulParam specifies the 32-bit optional parameter value

ulStatusFlag indicates the Local to Remote Flag number mask used to report the status of the command sent back from the control system.

Description:

This function will allow the Local CPU system to call a function on the Remote CPU. The *ulParam* variable is a single optional 32-bit parameter to pass to the function. The *ulFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32**. The *ulStatusFlag* parameter accepts any other one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**. The function returns **STATUS_PASS** if the command is successful or **STATUS_FAIL** if the request or status flags are unavailable.

Returns:

status of command (0=success, 1=error)

2.1.5.10 IPCLiteLtoRGetResult

Reads single word data result of Local to Remote IPC command

Prototype:

```
uint16_t
IPCLiteLtoRGetResult(void *pvData,
                    uint16_t usLength,
                    uint32_t ulStatusFlag)
```

Parameters:

pvData is a pointer to the 16/32-bit variable where the result data will be stored.

usLength designates 16- or 32-bit read.

ulStatusFlag indicates the Local to Remote CPU Flag number mask used to report the status of the command sent back from the Remote CPU. If a status flag was not used with the command call, set this parameter to 0.

Description:

Allows the caller to read the 16/32-bit data result of non-blocking IPC functions from the IPCRE-MOTEREPLY register if the status flag is cleared indicating the IPC command was successfully interpreted. If the status flag is not cleared, the command was not recognized, and the function will return STATUS_FAIL. To determine what data is read from a call to this function, see the descriptions of the non-blocking IPC functions. The *usLength* parameter accepts the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The *ulStatusFlag* parameter accepts any of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**. The function returns **STATUS_PASS** or **STATUS_FAIL**.

Returns:

status of command (0=success, 1=error)

2.1.5.11 IPCLiteLtoRSetBits

Sets the designated bits in a 16/32-bit data word at the remote CPU system address

Prototype:

```
uint16_t
IPCLiteLtoRSetBits(uint32_t ulFlag,
                  uint32_t ulAddress,
                  uint32_t ulMask,
                  uint16_t usLength,
                  uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Local to Remote IPC Flag number mask used to indicate a command is being sent.

ulAddress specifies the Remote address to write to.

ulMask specifies the 16/32-bit mask for bits which should be set at remote ulAddress. For 16-bit mask, only the lower 16-bits of ulMask are considered.

usLength specifies the length of the *ulMask* (1 = 16-bit, 2 = 32-bit).

ulStatusFlag indicates the Local to Remote Flag number mask used to report the status of the command sent back from the Remote system.

Description:

This function will allow the Local CPU system to set bits specified by the *usMask* variable in a 16/32-bit word on the Remote CPU system. The data word at /e ulAddress after the set bits

command is then read into the IPCREMOTEREPLY register. After calling this function, a call to [IPCLiteLtoRGetResult\(\)](#) will read the data value in the IPCREMOTEREPLY register into a 16/32-bit variable in the Local CPU application. The *usLength* parameter accepts the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The *ulStatusFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**. The function returns **STATUS_PASS** if the command is successful or **STATUS_FAIL** if the request or status flags are unavailable.

Returns:

status of command (0=success, 1=error)

2.1.5.12 IPCLiteLtoRSetBits_Protected

Sets the designated bits in a 16/32-bit write-protected data word at the Remote CPU system address

Prototype:

```
uint16_t  
IPCLiteLtoRSetBits_Protected(uint32_t ulFlag,  
                             uint32_t ulAddress,  
                             uint32_t ulMask,  
                             uint16_t usLength,  
                             uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Local to Remote IPC Flag number mask used to indicate a command is being sent.

ulAddress specifies the Remote CPU write-protected address to write to.

ulMask specifies the 16/32-bit mask for bits which should be set at Remote CPU ulAddress. For 16-bit mask, only the lower 16-bits of ulMask are considered.

usLength specifies the length of the *ulMask* (1 = 16-bit, 2 = 32-bit).

ulStatusFlag indicates the Local to Remote Flag number mask used to report the status of the command sent back from the Master system.

Description:

This function will allow the Local CPU system to set bits specified by the *usMask* variable in a write-protected 16/32-bit word on the REMote CPU system. The data word at /e ulAddress after the set bits command is then read into the IPCREMOTEREPLY register. After calling this function, a call to [IPCLiteLtoRGetResult\(\)](#) will read the data value in the IPCREMOTEREPLY register into a 16/32-bit variable in the Local application. The *usLength* parameter accepts the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The *ulStatusFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**. The function returns **STATUS_PASS** if the command is successful or **STATUS_FAIL** if the request or status flags are unavailable.

Returns:

status of command (0=success, 1=error)

2.1.5.13 IPCLiteReqMemAccess

Slave Requests Master R/W/Exe Access to Shared SARAM.

Prototype:

```
uint16_t
IPCLiteReqMemAccess (uint32_t ulFlag,
                    uint32_t ulMask,
                    uint16_t ulMaster,
                    uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Local to Remote IPC Flag number mask used to indicate a command is being sent.

ulMask specifies the 32-bit mask for the GSxMEMSEL RAM control register to indicate which GSx SARAM blocks the Slave is requesting master access to.

ulMaster specifies whether CPU1 or CPU2 should be the master of the GSx RAM.

ulStatusFlag indicates the Local to Remote Flag number mask used to report the status of the command sent back from the Master system.

Description:

This function will allow the slave CPU System to request slave or master mastership of any of the GSx Shared SARAM blocks. The *ulMaster* parameter accepts the following values: **IPC_GSX_CPU2_MASTER** or **IPC_GSX_CPU1_MASTER**. The *ulStatusFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**. The function returns **STATUS_PASS** if the command is successful or **STATUS_FAIL** if the request or status flags are unavailable.

Note:

This function calls the [*IPCLiteLtoRSetBits_Protected\(\)*](#) or the [*IPCLiteLtoRClearBits_Protected*](#) function, and therefore in order to process this function, the above 2 functions should be ready to be called on the master system to process this command.

Returns:

status of command (0=success, 1=error)

2.1.5.14 IPCLiteRtoLClearBits

Clears the designated bits in a 16/32-bit data word at Local CPU system address

Prototype:

```
void
IPCLiteRtoLClearBits (uint32_t ulFlag,
                    uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Remote to Local IPC Flag number mask used to indicate a command is being sent.

ulStatusFlag indicates the Remote to Local Flag number mask used to report the status of the command sent back from the control system.

Description:

This function will allow the Remote CPU system to clear bits specified by a mask variable in a 16/32-bit word on the Local CPU system, and then read back the word into the IPCLOCALREPLY register. The *ulFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32**, and the *ulStatusFlag* parameter accepts any other one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**.

2.1.5.15 void IPCLiteRtoLClearBits_Protected (uint32_t *ulFlag*, uint32_t *ulStatusFlag*)

Clears the designated bits in a 16/32-bit data word at the Local CPU system write-protected address

Parameters:

ulFlag specifies Remote to Local IPC Flag number mask used to indicate a command is being sent.

ulStatusFlag indicates the Remote to Local Flag number mask used to report the status of the command sent back from the control system.

This function will allow the Remote CPU system to clear bits specified by a mask variable in a 16/32-bit word on the Local CPU system, and then read back the word into the IPCLOCALREPLY register. The *ulFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32**, and the *ulStatusFlag* parameter accepts any other one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**.

2.1.5.16 IPCLiteRtoLDataRead

Reads either a 16- or 32-bit data word from the Local CPU system address

Prototype:

```
void
IPCLiteRtoLDataRead(uint32_t ulFlag,
                    uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Remote to Local IPC Flag number mask used to indicate a command is being sent.

ulStatusFlag indicates the Remote to Local Flag number mask used to report the status of the command sent back from the control system.

Description:

This function will allow the Remote CPU system to read 16/32-bit data from the Local CPU system. The *ulFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32**, and the *ulStatusFlag* parameter accepts any other one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**.

2.1.5.17 void IPCLiteRtoLDataWrite (uint32_t *ulFlag*, uint32_t *ulStatusFlag*)

Writes a 16/32-bit data word to Local CPU system address

Parameters:

ulFlag specifies Remote to Local IPC Flag number mask used to indicate a command is being sent.

ulStatusFlag indicates the Remote to Local Flag number mask used to report the status of the command sent back from the control system.

This function will allow the Remote CPU system to write a 16/32-bit word to an address on the Local CPU system. The *ulFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32**, and the *ulStatusFlag* parameter accepts any other one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**.

2.1.5.18 IPCLiteRtoLDataWrite_Protected

Writes a 16/32-bit data word to a write-protected Local CPU system address

Prototype:

```
void  
IPCLiteRtoLDataWrite_Protected(uint32_t ulFlag,  
                               uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Remote to Local IPC Flag number mask used to indicate a command is being sent.

ulStatusFlag indicates the Remote to Local Flag number mask used to report the status of the command sent back from the control system.

Description:

This function will allow the Remote CPU system to write a 16/32-bit word to an address on the Local CPU system. The *ulFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32**, and the *ulStatusFlag* parameter accepts any other one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**.

2.1.5.19 void IPCLiteRtoLFunctionCall (uint32_t ulFlag, uint32_t ulStatusFlag)

Calls a Local CPU function with a single optional parameter and return value.

Parameters:

ulFlag specifies Remote to Local IPC Flag number mask used to indicate a command is being sent.

ulStatusFlag indicates the Remote to Local Flag number mask used to report the status of the command sent back from the control system.

This function will allow the Remote CPU system to call a Local CPU function with a single optional parameter and places an optional return value in the IPCLOCALREPLY register. The *ulFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32**, and the *ulStatusFlag* parameter accepts any other one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**.

2.1.5.20 IPCLiteRtoLSetBits

Sets the designated bits in a 16/32-bit data word at the Local CPU system address

Prototype:

```
void  
IPCLiteRtoLSetBits(uint32_t ulFlag,  
                   uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Remote to Local IPC Flag number mask used to indicate a command is being sent.

ulStatusFlag indicates the Remote to Local Flag number mask used to report the status of the command sent back from the control system.

Description:

This function will allow the Remote CPU system to set bits specified by a mask variable in a 16/32-bit word on the Local CPU system, and then read back the word into the IPCLOCALREPLY register. The *ulFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32**, and the *ulStatusFlag* parameter accepts any other one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**.

2.1.5.21 void IPCLiteRtoLSetBits_Protected (uint32_t ulFlag, uint32_t ulStatusFlag)

Sets the designated bits in a 16-bit data word at the Local CPU system write-protected address

Parameters:

ulFlag specifies Remote to Local IPC Flag number mask used to indicate a command is being sent.

ulStatusFlag indicates the Remote to Local Flag number mask used to report the status of the command sent back from the control system.

This function will allow the Remote CPU system to set bits specified by a mask variable in a write-protected 16/32-bit word on the Local CPU system, and then read back the word into the IPCLOCALREPLY register. The *ulFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32**, and the *ulStatusFlag* parameter accepts any other one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**.

2.1.5.22 IPCLtoRBlockRead

Sends the command to read a block of data from remote CPU system address

Prototype:

```
uint16_t  
IPCLtoRBlockRead(volatile tIpcController *psController,  
                 uint32_t ulAddress,  
                 uint32_t ulShareAddress,  
                 uint16_t usLength,  
                 uint16_t bBlock,  
                 uint32_t ulResponseFlag)
```

Parameters:

psController specifies the address of a *tIpcController* instance used to store information about the "Put" and "Get" circular buffers and their respective indexes.

ulAddress specifies the remote CPU memory block starting address to read from.

ulShareAddress specifies the local CPU shared memory address the read block will read to.
usLength designates the block size in 16-bit words.

bBlock specifies whether to allow function to block until PutBuffer has a slot (1= wait until slot free, 0 = exit with STATUS_FAIL if no slot).

ulResponseFlag indicates the local CPU to remote CPU Flag number mask used to report when the read block data is available starting at /e ulShareAddress. (*ulResponseFlag* MUST use IPC flags 17-32, and not 1-16)

Description:

This function will allow the local CPU system to send a read block command to the remote CPU system and set a ResponseFlag to track the status of the read. The remote CPU system will process the read and place the data in shared memory at the location specified in the *ulShareAddress* parameter and then clear the ResponseFlag, indicating that the block is ready. The *bBlock* parameter can be one of the following values: **ENABLE_BLOCKING** or **DISABLE_BLOCKING**. The *ulResponseFlag* parameter can be any single one of the flags **IPC_FLAG16 - IPC_FLAG31** or **NO_FLAG**.

Returns:

status of command (**STATUS_PASS** =success, **STATUS_FAIL** = error because PutBuffer was full, command was not sent)

2.1.5.23 IPCLtoRBlockWrite

Writes a block of data to remote CPU system address

Prototype:

```
uint16_t
IPCLtoRBlockWrite(volatile tIpcController *psController,
                  uint32_t ulAddress,
                  uint32_t ulShareAddress,
                  uint16_t usLength,
                  uint16_t usWordLength,
                  uint16_t bBlock)
```

Parameters:

psController specifies the address of a *tIpcController* instance used to store information about the "Put" and "Get" circular buffers and their respective indexes.

ulAddress specifies the remote CPU memory block starting address to write to.

ulShareAddress specifies the local CPU shared memory address where data to write from resides.

usLength designates the block size in 16- or 32-bit words (depends on *usWordLength*).

usWordLength designates the word size (16-bits = 1 or 32-bits = 2).

bBlock specifies whether to allow function to block until PutBuffer has a slot (1= wait until slot free, 0 = exit with STATUS_FAIL if no slot).

Description:

This function will allow the local CPU system to write a block of data to the remote CPU system starting from the location specified by the *ulAddress* parameter. Prior to calling this function, the local CPU system code should place the data to write in shared memory starting at /e ulShareAddress. The *usWordLength* parameter can be one of the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The *bBlock* parameter can be one of the

following values: **ENABLE_BLOCKING** or **DISABLE_BLOCKING**. The *ulResponseFlag* parameter can be any single one of the flags **IPC_FLAG16** - **IPC_FLAG31** or **NO_FLAG**.

Note:

If the shared SARAM blocks are used to pass the RAM block between the processors, the `IPCReqMemAccess()` function must be called first in order to give the slave CPU write access to the shared memory block(s).

Returns:

status of command (**STATUS_PASS** = success, **STATUS_FAIL** = error because PutBuffer was full, command was not sent)

2.1.5.24 IPLtoRBlockWrite_Protected

Writes a block of data to a write-protected remote CPU system address

Prototype:

```
uint16_t  
IPLtoRBlockWrite_Protected(volatile tIpcController *psController,  
                           uint32_t ulAddress,  
                           uint32_t ulShareAddress,  
                           uint16_t usLength,  
                           uint16_t usWordLength,  
                           uint16_t bBlock)
```

Parameters:

psController specifies the address of a *tIpcController* instance used to store information about the "Put" and "Get" circular buffers and their respective indexes.

ulAddress specifies the write-protected remote CPU block starting address to write to.

ulShareAddress specifies the local CPU shared memory address where data to write from resides.

usLength designates the block size in 16- or 32-bit words (depends on *usWordLength*).

usWordLength designates the word size (16-bits = 1 or 32-bits = 2).

bBlock specifies whether to allow function to block until PutBuffer has a slot (1= wait until slot free, 0 = exit with **STATUS_FAIL** if no slot).

Description:

This function will allow the local CPU system to write a block of data to a write-protected region on the remote CPU system starting from the location specified by the *ulAddress* parameter. Prior to calling this function, the local CPU system code should place the data to write in shared memory starting at *ulShareAddress*. The *usWordLength* parameter can be one of the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The *bBlock* parameter can be one of the following values: **ENABLE_BLOCKING** or **DISABLE_BLOCKING**. The *ulResponseFlag* parameter can be any single one of the flags **IPC_FLAG16** - **IPC_FLAG31** or **NO_FLAG**.

Note:

If the shared SARAM blocks are used to pass the RAM block between the processors, the `IPCReqMemAccess()` function must be called first in order to give the the slave CPU write access to the shared memory block(s).

Returns:

status of command (**STATUS_PASS** =success, **STATUS_FAIL** = error because PutBuffer was full, command was not sent)

2.1.5.25 IPCLtoRClearBits

Clears the designated bits in a 16-bit data word at the remote CPU system address

Prototype:

```
uint16_t
IPCLtoRClearBits(volatile tIpcController *psController,
                  uint32_t ulAddress,
                  uint32_t ulMask,
                  uint16_t usLength,
                  uint16_t bBlock)
```

Parameters:

psController specifies the address of a *tIpcController* instance used to store information about the "Put" and "Get" circular buffers and their respective indexes.

ulAddress specifies the remote CPU address to write to

ulMask specifies the 16/32-bit mask for bits which should be cleared at *ulAddress*. 16-bit masks should fill the lower 16-bits (upper 16-bits will be all 0x0000).

usLength specifies the length of the bit mask (1=16-bits, 2=32-bits)

bBlock specifies whether to allow function to block until PutBuffer has a slot (1= wait until slot free, 0 = exit with **STATUS_FAIL** if no slot).

Description:

This function will allow the local CPU system to clear bits specified by the *ulMask* variable in a 16/32-bit word on the remote CPU system. The *usLength* parameter can be one of the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The *bBlock* parameter can be one of the following values: **ENABLE_BLOCKING** or **DISABLE_BLOCKING**.

Returns:

status of command (**STATUS_PASS** =success, **STATUS_FAIL** = error because PutBuffer was full, command was not sent)

2.1.5.26 IPCLtoRClearBits_Protected

Clears the designated bits in a 16-bit write-protected data word at remote CPU system address

Prototype:

```
uint16_t
IPCLtoRClearBits_Protected(volatile tIpcController *psController,
                             uint32_t ulAddress,
                             uint32_t ulMask,
                             uint16_t usLength,
                             uint16_t bBlock)
```

Parameters:

psController specifies the address of a *tIpcController* instance used to store information about the "Put" and "Get" circular buffers and their respective indexes.

ulAddress specifies the secondary CPU address to write to

ulMask specifies the 16/32-bit mask for bits which should be cleared at *ulAddress*. 16-bit masks should fill the lower 16-bits (upper 16-bits will be all 0x0000).

usLength specifies the length of the bit mask (1=16-bits, 2=32-bits)

bBlock specifies whether to allow function to block until PutBuffer has a slot (1= wait until slot free, 0 = exit with STATUS_FAIL if no slot).

Description:

This function will allow the local CPU system to set bits specified by the *ulMask* variable in a write-protected 16/32-bit word on the remote CPU system. The *usLength* parameter can be one of the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The *bBlock* parameter can be one of the following values: **ENABLE_BLOCKING** or **DISABLE_BLOCKING**.

Returns:

status of command (**STATUS_PASS** =success, **STATUS_FAIL** = error because PutBuffer was full, command was not sent)

2.1.5.27 IPCLtoRDataRead

Sends a command to read either a 16- or 32-bit data word from the remote CPU

Prototype:

```
uint16_t
IPCLtoRDataRead(volatile tIpcController *psController,
                 uint32_t ulAddress,
                 void *pvData,
                 uint16_t usLength,
                 uint16_t bBlock,
                 uint32_t ulResponseFlag)
```

Parameters:

psController specifies the address of a *tIpcController* instance used to store information about the "Put" and "Get" circular buffers and their respective indexes.

ulAddress specifies the remote CPU address to read from

pvData is a pointer to the 16/32-bit variable where read data will be stored.

usLength designates 16- or 32-bit read (1 = 16-bit, 2 = 32-bit)

bBlock specifies whether to allow function to block until PutBuffer has a slot (1= wait until slot free, 0 = exit with STATUS_FAIL if no slot).

ulResponseFlag indicates the remote CPU to the local CPU Flag number mask used to report when the read data is available at *pvData*. (*ulResponseFlag* MUST use IPC flags 17-32, and not 1-16)

Description:

This function will allow the local CPU system to send a 16/32-bit data read command to the remote CPU system and set a ResponseFlag to track the status of the read. The remote CPU will respond with a DataWrite command which will place the data in the local CPU address pointed to by *pvData*. When the local CPU receives the DataWrite command and writes the read data into **pvData* location, it will clear the ResponseFlag, indicating to the rest of the system that the data is ready. The *usLength* parameter can be one of the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The *bBlock* parameter can be one of

the following values: **ENABLE_BLOCKING** or **DISABLE_BLOCKING**. The *ulResponseFlag* parameter can be any single one of the flags **IPC_FLAG16** - **IPC_FLAG31** or **NO_FLAG**.

Returns:

status of command (**STATUS_PASS** = success, **STATUS_FAIL** = error because PutBuffer was full, command was not sent)

2.1.5.28 IPLtoRDataRead_Protected

Sends the command to read either a 16- or 32-bit data word from remote CPU system address to a write-protected local CPU address.

Prototype:

```
uint16_t  
IPLtoRDataRead_Protected(volatile tipcController *psController,  
                          uint32_t ulAddress,  
                          void *pvData,  
                          uint16_t usLength,  
                          uint16_t bBlock,  
                          uint32_t ulResponseFlag)
```

Parameters:

psController specifies the address of a *tipcController* instance used to store information about the "Put" and "Get" circular buffers and their respective indexes.

ulAddress specifies the remote CPU address to read from

pvData is a pointer to the 16/32-bit variable where read data will be stored.

usLength designates 16- or 32-bit read (1 = 16-bit, 2 = 32-bit)

bBlock specifies whether to allow function to block until PutBuffer has a slot (1= wait until slot free, 0 = exit with STATUS_FAIL if no slot).

ulResponseFlag indicates the local CPU to remote CPU Flag number mask used to report when the read data is available at pvData. (*ulResponseFlag* MUST use IPC flags 17-32, and not 1-16)

Description:

This function will allow the local CPU system to send a 16/32-bit data read command to the remote CPU system and set a ResponseFlag to track the status of the read. The remote CPU system will respond with a DataWrite command which will place the data in the local CPU address pointed to by *pvData*. When the local CPU receives the DataWrite command and writes the read data into *pvData* location, it will clear the ResponseFlag, indicating to the rest of the system that the data is ready. The *usLength* parameter can be one of the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The *bBlock* parameter can be one of the following values: **ENABLE_BLOCKING** or **DISABLE_BLOCKING**. The *ulResponseFlag* parameter can be any single one of the flags **IPC_FLAG16** - **IPC_FLAG31** or **NO_FLAG**.

Returns:

status of command (**STATUS_PASS** = success, **STATUS_FAIL** = error because PutBuffer was full, command was not sent)

2.1.5.29 IPCLtoRDataWrite

Writes a 16/32-bit data word to the remote CPU system address

Prototype:

```
uint16_t  
IPCLtoRDataWrite(volatile tIpcController *psController,  
                  uint32_t ulAddress,  
                  uint32_t ulData,  
                  uint16_t usLength,  
                  uint16_t bBlock,  
                  uint32_t ulResponseFlag)
```

Parameters:

psController specifies the address of a *tIpcController* instance used to store information about the "Put" and "Get" circular buffers and their respective indexes.

ulAddress specifies the remote cpu address to write to

ulData specifies the 16/32-bit word which will be written. For 16-bit words, only the lower 16-bits of ulData will be considered by the master system.

usLength is the length of the word to write (1 = 16-bits, 2 = 32-bits)

bBlock specifies whether to allow function to block until PutBuffer has a slot (1= wait until slot free, 0 = exit with STATUS_FAIL if no slot).

ulResponseFlag is used to pass the *ulResponseFlag* back to the remote cpu only when this function is called in response to *IPCMtoCDataRead()*. Otherwise, set to 0.

Description:

This function will allow the local CPU system to write a 16/32-bit word via the *ulData* variable to an address on the remote CPU system. The *usLength* parameter can be one of the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The *bBlock* parameter can be one of the following values: **ENABLE_BLOCKING** or **DISABLE_BLOCKING**. The *ulResponseFlag* parameter can be any single one of the flags **IPC_FLAG16** - **IPC_FLAG31** or **NO_FLAG**.

Returns:

status of command (**STATUS_PASS** =success, **STATUS_FAIL** = error because PutBuffer was full, command was not sent)

2.1.5.30 IPCLtoRDataWrite_Protected

Writes a 16/32-bit data word to a write-protected remote CPU system address

Prototype:

```
uint16_t  
IPCLtoRDataWrite_Protected(volatile tIpcController *psController,  
                            uint32_t ulAddress,  
                            uint32_t ulData,  
                            uint16_t usLength,  
                            uint16_t bBlock,  
                            uint32_t ulResponseFlag)
```


Parameters:

psController specifies the address of a [tlpcController](#) instance used to store information about the "Put" and "Get" circular buffers and their respective indexes.

ulAddress specifies the write-protected remote CPU address to write to

ulData specifies the 16/32-bit word which will be written. For 16-bit words, only the lower 16-bits of ulData will be considered by the master system.

usLength is the length of the word to write (1 = 16-bits, 2 = 32-bits)

bBlock specifies whether to allow function to block until PutBuffer has a slot (1= wait until slot free, 0 = exit with STATUS_FAIL if no slot).

ulResponseFlag is used to pass the *ulResponseFlag* back to the remote CPU only when this function is called in response to *IPCMtoCDataRead()*. Otherwise, set to 0.

Description:

This function will allow the local CPU system to write a 16/32-bit word via the *ulData* variable to a write-protected address on the remote CPU system. The *usLength* parameter can be one of the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The *bBlock* parameter can be one of the following values: **ENABLE_BLOCKING** or **DISABLE_BLOCKING**. The *ulResponseFlag* parameter can be any single one of the flags **IPC_FLAG16** - **IPC_FLAG31** or **NO_FLAG**.

Returns:

status of command (**STATUS_PASS** =success, **STATUS_FAIL** = error because PutBuffer was full, command was not sent)

2.1.5.31 IPCLtoRFlagBusy

Determines whether the given IPC flags are busy or not.

Prototype:

```
Uint16  
IPCLtoRFlagBusy(uint32_t ulFlags)
```

Parameters:

ulFlags specifies Local to Remote IPC Flag number masks to check the status of.

Description:

Allows the caller to determine whether the designated IPC flags are available for further control to master system communication. If **0** is returned, then all designated tasks have completed and are available. The *ulFlags* parameter can be any of the IPC flag values: **IPC_FLAG0** - **IPC_FLAG31**.

Returns:

Returns **1** if the IPC flags are busy or **0** if designated IPC flags are free.

2.1.5.32 IPCLtoRFlagClear

Local CPU Clears Local to Remote IPC Flag

Prototype:

```
void  
IPCLtoRFlagClear(uint32_t ulFlags)
```

Parameters:

ulFlags specifies the IPC flag mask for flags being set.

Description:

This function will allow the Local CPU system to set the designated IPC flags to send to the Remote CPU system. The *ulFlags* parameter can be any of the IPC flag values: **IPC_FLAG0** - **IPC_FLAG31**.

Returns:

None.

2.1.5.33 IPCLtoRFlagSet

Local CPU Sets Local to Remote IPC Flag

Prototype:

```
void  
IPCLtoRFlagSet (uint32_t ulFlags)
```

Parameters:

ulFlags specifies the IPC flag mask for flags being set.

Description:

This function will allow the Local CPU system to set the designated IPC flags to send to the Remote CPU system. The *ulFlags* parameter can be any of the IPC flag values: **IPC_FLAG0** - **IPC_FLAG31**.

Returns:

None.

2.1.5.34 IPCLtoRFunctionCall

Calls remote CPU function with 1 optional parameter .

Prototype:

```
uint16_t  
IPCLtoRFunctionCall (volatile tIpcController *psController,  
                     uint32_t ulAddress,  
                     uint32_t ulParam,  
                     uint16_t bBlock)
```

Parameters:

psController specifies the address of a *tIpcController* instance used to store information about the "Put" and "Get" circular buffers and their respective indexes.

ulAddress specifies the remote CPU function address

ulParam specifies the 32-bit optional parameter value. If not used, this can be a dummy value.

bBlock specifies whether to allow function to block until PutBuffer has a slot (1= wait until slot free, 0 = exit with STATUS_FAIL if no slot).

Description:

This function will allow the local CPU system to call a function on the remote CPU. The *ulParam* variable is a single optional 32-bit parameter to pass to the function. The *bBlock* parameter can be one of the following values: **ENABLE_BLOCKING** or **DISABLE_BLOCKING**.

Returns:

status of command (**STATUS_PASS** = success, **STATUS_FAIL** = error because PutBuffer was full, command was not sent)

2.1.5.35 IPCLtoRSendMessage

Sends generic message to remote CPU system

Prototype:

```
uint16_t
IPCLtoRSendMessage(volatile tIpcController *psController,
                    uint32_t ulCommand,
                    uint32_t ulAddress,
                    uint32_t ulDataW1,
                    uint32_t ulDataW2,
                    uint16_t bBlock)
```

Parameters:

psController specifies the address of a *tIpcController* instance used to store information about the "Put" and "Get" circular buffers and their respective indexes.

ulCommand specifies 32-bit command word to insert into message.

ulAddress specifies 32-bit address word to insert into message.

ulDataW1 specifies 1st 32-bit data word to insert into message.

ulDataW2 specifies 2nd 32-bit data word to insert into message.

bBlock specifies whether to allow function to block until PutBuffer has a slot (1= wait until slot free, 0 = exit with STATUS_FAIL if no slot).

Description:

This function will allow the local CPU system to send a generic message to the remote CPU system. Note that the user should create a corresponding function on the remote CPU side to interpret/use the message or fill parameters in such a way that the existing IPC drivers can recognize the command and properly process the message. The *bBlock* parameter can be one of the following values: **ENABLE_BLOCKING** or **DISABLE_BLOCKING**.

Returns:

status of command (**STATUS_PASS** = success, **STATUS_FAIL** = error because PutBuffer was full, command was not sent)

2.1.5.36 IPCLtoRSetBits

Sets the designated bits in a 16-bit data word at the remote CPU system address

Prototype:

```
uint16_t
IPCLtoRSetBits(volatile tIpcController *psController,
```

```
uint32_t ulAddress,  
uint32_t ulMask,  
uint16_t usLength,  
uint16_t bBlock)
```

Parameters:

psController specifies the address of a [tlpcController](#) instance used to store information about the "Put" and "Get" circular buffers and their respective indexes.

ulAddress specifies the remote CPU address to write to

ulMask specifies the 16/32-bit mask for bits which should be set at *ulAddress*. 16-bit masks should fill the lower 16-bits (upper 16-bits will be all 0x0000).

usLength specifies the length of the bit mask (1=16-bits, 2=32-bits)

bBlock specifies whether to allow function to block until PutBuffer has a slot (1= wait until slot free, 0 = exit with STATUS_FAIL if no slot).

Description:

This function will allow the local CPU system to set bits specified by the *ulMask* variable in a 16/32-bit word on the remote CPU system. The *usLength* parameter can be one of the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The *bBlock* parameter can be one of the following values: **ENABLE_BLOCKING** or **DISABLE_BLOCKING**.

Returns:

status of command (**STATUS_PASS** =success, **STATUS_FAIL** = error because PutBuffer was full, command was not sent)

2.1.5.37 IPCLtoRSetBits_Protected

Sets the designated bits in a 16-bit write-protected data word at the remote CPU system address

Prototype:

```
uint16_t  
IPCLtoRSetBits_Protected(volatile tlpcController *psController,  
                          uint32_t ulAddress,  
                          uint32_t ulMask,  
                          uint16_t usLength,  
                          uint16_t bBlock)
```

Parameters:

psController specifies the address of a [tlpcController](#) instance used to store information about the "Put" and "Get" circular buffers and their respective indexes.

ulAddress specifies the remote CPU address to write to

ulMask specifies the 16/32-bit mask for bits which should be set at *ulAddress*. 16-bit masks should fill the lower 16-bits (upper 16-bits will be all 0x0000).

usLength specifies the length of the bit mask (1=16-bits, 2=32-bits)

bBlock specifies whether to allow function to block until PutBuffer has a slot (1= wait until slot free, 0 = exit with STATUS_FAIL if no slot).

Description:

This function will allow the local CPU system to set bits specified by the *ulMask* variable in a write-protected 16/32-bit word on the remote CPU system. The *usLength* parameter can be one of the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**.

The *bBlock* parameter can be one of the following values: **ENABLE_BLOCKING** or **DISABLE_BLOCKING**.

Returns:

status of command (**STATUS_PASS** = success, **STATUS_FAIL** = error because PutBuffer was full, command was not sent)

2.1.5.38 IpcPut

Writes a message into the PutBuffer.

Prototype:

```
uint16_t  
IpcPut(volatile tIpcController *psController,  
        tIpcMessage *psMessage,  
        uint16_t bBlock)
```

Parameters:

psController specifies the address of a *tIpcController* instance used to store information about the "Put" and "Get" circular buffers and their respective indexes.

psMessage specifies the address of the *tIpcMessage* instance to be written to PutBuffer.

bBlock specifies whether to allow function to block until PutBuffer has a free slot (1= wait until free spot available, 0 = exit with STATUS_FAIL if no free slot).

Description:

This function checks if there is a free slot in the PutBuffer. If so, it puts the message pointed to by *psMessage* into the free slot and increments the WriteIndex. Then it sets the appropriate IPC interrupt flag specified by *psController->usPutFlag*. The *bBlock* parameter can be one of the following values: **ENABLE_BLOCKING** or **DISABLE_BLOCKING**.

Returns:

STATUS_FAIL if PutBuffer is full. **STATUS_PASS** if Put occurs successfully.

2.1.5.39 IPCRtoLBlockRead

Reads a block of data from a remote CPU system address and stores into shared RAM

Prototype:

```
void  
IPCRtoLBlockRead(tIpcMessage *psMessage)
```

Parameters:

psMessage specifies the pointer to the message received from the remote CPU system.

Description:

This function will respond to the remote CPU system request to read a block of data from the local control system, by reading the data and placing that data into the shared RAM location specified in *psMessage*.

Returns:

None.

2.1.5.40 IPCRtoLBlockWrite

Writes a block of data to a local CPU system address from shared RAM

Prototype:

```
void  
IPCRtoLBlockWrite(tIpcMessage *psMessage)
```

Parameters:

psMessage specifies the pointer to the message received from the remote CPU system.

Description:

This function will write a block of data to an address on the local CPU system. The data is first written by the remote CPU to shared RAM. This function reads the shared RAM location, word size (16- or 32-bit), and block size from *psMessage* and writes the block to the local address specified in *psMessage*.

Returns:

None.

2.1.5.41 IPCRtoLBlockWrite_Protected

Writes a block of data to a local CPU system write-protected address from shared RAM

Prototype:

```
void  
IPCRtoLBlockWrite_Protected(tIpcMessage *psMessage)
```

Parameters:

psMessage specifies the pointer to the message received from the remote CPU system.

Description:

This function will write a block of data to a write-protected group of addresses on the local CPU system. The data is first written by the remote CPU to shared RAM. This function reads the shared RAM location, word size (16- or 32-bit), and block size from *psMessage* and writes the block to the local address specified in *psMessage*.

Returns:

None.

2.1.5.42 IPCRtoLClearBits

Clears the designated bits in a 32-bit data word at a local CPU system address

Prototype:

```
void  
IPCRtoLClearBits(tIpcMessage *psMessage)
```

Parameters:

psMessage specifies the pointer to the message received from the remote CPU system.

Description:

This function will allow the remote CPU system to clear the bits in a 16/32-bit word on the local CPU system via a local CPU address and mask passed in via the *psMessage*.

Returns:

None.

2.1.5.43 IPCRtoLClearBits_Protected

Clears the designated bits in a write-protected 16/32-bit data word at a local CPU system address

Prototype:

```
void  
IPCRtoLClearBits_Protected(tIpcMessage *psMessage)
```

Parameters:

psMessage specifies the pointer to the message received from the remote CPU system.

Description:

This function will allow the secondary CPU system to clear the bits in a 16/32-bit write-protected word on the local CPU system via a local CPU address and mask passed in via the *psMessage*.

Returns:

None.

2.1.5.44 IPCRtoLDataRead

Responds to 16/32-bit data word read command from remote CPU system.

Prototype:

```
void  
IPCRtoLDataRead(volatile tIpcController *psController,  
                 tIpcMessage *psMessage,  
                 uint16_t bBlock)
```

Parameters:

psController specifies the address of a *tIpcController* instance used to store information about the "Put" and "Get" circular buffers and their respective indexes.

psMessage specifies the pointer to the message received from the remote CPU system.

bBlock specifies whether to allow function to block until PutBuffer has a slot (1= wait until slot free, 0 = exit with STATUS_FAIL if no slot).

Description:

This function will allow the remote CPU system to read a 16/32-bit data word at the local CPU address specified in /e *psMessage*, and send a Write command with the read data back to the local CPU system. It will also send the Response Flag used to track the read back to the remote CPU. The *bBlock* parameter can be one of the following values: **ENABLE_BLOCKING** or **DISABLE_BLOCKING**.

Returns:

None.

2.1.5.45 IPCRtoLDataRead_Protected

Responds to 16/32-bit data word read command from remote CPU system. to read into a write-protected word on the remote CPU system.

Prototype:

```
void  
IPCRtoLDataRead_Protected(volatile tIpcController *psController,  
                           tIpcMessage *psMessage,  
                           uint16_t bBlock)
```

Parameters:

psController specifies the address of a *tIpcController* instance used to store information about the "Put" and "Get" circular buffers and their respective indexes.

psMessage specifies the pointer to the message received from the remote CPU system.

bBlock specifies whether to allow function to block until PutBuffer has a slot (1= wait until slot free, 0 = exit with STATUS_FAIL if no slot).

Description:

This function will allow the remote CPU system to read a 16/32-bit data word at the local CPU address specified in /e *psMessage*, and send a Write Protected command with the read data back to the remote CPU system at a write protected address. It will also send the Response Flag used to track the read back to the remote CPU. The *bBlock* parameter can be one of the following values: **ENABLE_BLOCKING** or **DISABLE_BLOCKING**.

Returns:

None.

2.1.5.46 IPCRtoLDataWrite

Responds to 16/32-bit data word write command the remote CPU system

Prototype:

```
void  
IPCRtoLDataWrite(tIpcMessage *psMessage)
```

Parameters:

psMessage specifies the pointer to the message received from remote CPU system which includes the 16/32-bit data word to write to the local CPU address.

Description:

This function will allow the local CPU system to write a 16/32-bit word received from the remote CPU system to the address indicated in *psMessage*. In the event that the IPC_DATA_WRITE command was received as a result of an IPC_DATA_READ command, this function will also clear the IPC response flag tracking the read so other threads in the local CPU system will be aware that the data is ready.

Returns:

None.

2.1.5.47 IPCRtoLDataWrite_Protected

Responds to 16/32-bit write-protected data word write command from secondary CPU system

Prototype:

```
void  
IPCRtoLDataWrite_Protected(tIpcMessage *psMessage)
```

Parameters:

psMessage specifies the pointer to the message received from the secondary CPU system which includes the 16/32-bit data word to write to the local CPU address.

Description:

This function will allow the local CPU system to write a 16/32-bit word received from the secondary CPU system to the write-protected address indicated in *psMessage*. In the event that the IPC_DATA_WRITE_PROTECTED command was received as a result of an IPC_DATA_READ_PROTECTED command, this function will also clear the IPC response flag tracking the read so other threads in the local CPU will be aware that the data is ready.

Returns:

None.

2.1.5.48 IPCRtoLFlagAcknowledge

Local CPU Acknowledges Remote to Local IPC Flag.

Prototype:

```
void  
IPCRtoLFlagAcknowledge(uint32_t ulFlags)
```

Parameters:

ulFlags specifies the IPC flag mask for flags being acknowledged.

Description:

This function will allow the Local CPU system to acknowledge/clear the IPC flag set by the Remote CPU system. The *ulFlags* parameter can be any of the IPC flag values: **IPC_FLAG0** - **IPC_FLAG31**.

Returns:

None.

2.1.5.49 IPCRtoLFlagBusy

Determines whether the given Remote to Local IPC flags are busy or not.

Prototype:

```
Uint16  
IPCRtoLFlagBusy(uint32_t ulFlags)
```

Parameters:

ulFlags specifies Remote to Local IPC Flag number masks to check the status of.

Description:

Allows the caller to determine whether the designated IPC flags are pending. The *ulFlags* parameter can be any of the IPC flag values: **IPC_FLAG0** - **IPC_FLAG31**.

Returns:

Returns **1** if the IPC flags are busy or **0** if designated IPC flags are free.

2.1.5.50 IPCRtoLFunctionCall

Calls a local function with a single optional parameter.

Prototype:

```
void  
IPCRtoLFunctionCall (tIpcMessage *psMessage)
```

Parameters:

psMessage specifies the pointer to the message received from the remote CPU system.

Description:

This function will allow the remote CPU system to call a local CPU function with a single optional parameter. There is no return value from the executed function.

Returns:

None.

2.1.5.51 IPCRtoLSetBits

Sets the designated bits in a 16/32-bit data word at a local CPU system address

Prototype:

```
void  
IPCRtoLSetBits (tIpcMessage *psMessage)
```

Parameters:

psMessage specifies the pointer to the message received from the remote CPU system.

Description:

This function will allow the remote CPU system to set the bits in a 16/32-bit word on the local CPU system via a local CPU address and mask passed in via the *psMessage*.

Returns:

None.

2.1.5.52 IPCRtoLSetBits_Protected

Sets the designated bits in a 16/32-bit write-protected data word at a local CPU system address

Prototype:

```
void  
IPCRtoLSetBits_Protected (tIpcMessage *psMessage)
```

Parameters:

psMessage specifies the pointer to the message received from the remote CPU system.

Description:

This function will allow the remote CPU system to set the bits in a write-protected 16/32-bit word on the local CPU system via a local CPU address and mask passed in via the *psMessage*.

Returns:

None

2.2 IPC-Lite API Drivers

Functions

- `uint16_t IPCLiteLtoRClearBits` (`uint32_t ulFlag`, `uint32_t ulAddress`, `uint32_t ulMask`, `uint16_t usLength`, `uint32_t ulStatusFlag`)
- `uint16_t IPCLiteLtoRClearBits_Protected` (`uint32_t ulFlag`, `uint32_t ulAddress`, `uint32_t ulMask`, `uint16_t usLength`, `uint32_t ulStatusFlag`)
- `uint16_t IPCLiteLtoRDataRead` (`uint32_t ulFlag`, `uint32_t ulAddress`, `uint16_t usLength`, `uint32_t ulStatusFlag`)
- `uint16_t IPCLiteLtoRDataWrite` (`uint32_t ulFlag`, `uint32_t ulAddress`, `uint32_t ulData`, `uint16_t usLength`, `uint32_t ulStatusFlag`)
- `uint16_t IPCLiteLtoRDataWrite_Protected` (`uint32_t ulFlag`, `uint32_t ulAddress`, `uint32_t ulData`, `uint16_t usLength`, `uint32_t ulStatusFlag`)
- `uint16_t IPCLiteLtoRFunctionCall` (`uint32_t ulFlag`, `uint32_t ulAddress`, `uint32_t ulParam`, `uint32_t ulStatusFlag`)
- `uint16_t IPCLiteLtoRGetResult` (`void *pvData`, `uint16_t usLength`, `uint32_t ulStatusFlag`)
- `uint16_t IPCLiteLtoRSetBits` (`uint32_t ulFlag`, `uint32_t ulAddress`, `uint32_t ulMask`, `uint16_t usLength`, `uint32_t ulStatusFlag`)
- `uint16_t IPCLiteLtoRSetBits_Protected` (`uint32_t ulFlag`, `uint32_t ulAddress`, `uint32_t ulMask`, `uint16_t usLength`, `uint32_t ulStatusFlag`)
- `uint16_t IPCLiteReqMemAccess` (`uint32_t ulFlag`, `uint32_t ulMask`, `uint16_t ulMaster`, `uint32_t ulStatusFlag`)
- `void IPCLiteRtoLClearBits` (`uint32_t ulFlag`, `uint32_t ulStatusFlag`)
- `void IPCLiteRtoLClearBits_Protected` (`uint32_t ulFlag`, `uint32_t ulStatusFlag`)
- `void IPCLiteRtoLDataRead` (`uint32_t ulFlag`, `uint32_t ulStatusFlag`)
- `void IPCLiteRtoLDataWrite` (`uint32_t ulFlag`, `uint32_t ulStatusFlag`)
- `void IPCLiteRtoLDataWrite_Protected` (`uint32_t ulFlag`, `uint32_t ulStatusFlag`)
- `void IPCLiteRtoLFunctionCall` (`uint32_t ulFlag`, `uint32_t ulStatusFlag`)
- `void IPCLiteRtoLSetBits` (`uint32_t ulFlag`, `uint32_t ulStatusFlag`)
- `void IPCLiteRtoLSetBits_Protected` (`uint32_t ulFlag`, `uint32_t ulStatusFlag`)

2.2.1 Detailed Description

The IPC-Lite drivers utilize the IPC registers to send command messages one at a time via a single IPC interrupt. No additional shared memory/message RAM communication is required.

Only one command can be sent and processed while the IPC interrupt flag is set. Once the IPC interrupt flag is acknowledged (cleared), another command can be sent. Only a single

ISR can be used with the IPC-Lite drivers at a time. The same IPC APIs are shared by both CPU1 and CPU2 processors. As a result all the APIs use the acronyms "LtoR" or "RtoL"

to represent "Local To Remote" and "Remote to Local" CPU access. For example if the function

IPCLiteLtoRDataWrite is called from CPU1, CPU1 would be the local whereas CPU2 would be the remote.

2.2.2 Usage and Examples

The IPC-Lite drivers operate as follows in a 2 processor device where X and Y are the arbitrary processor identifiers:

1. Processor X's application code calls IPC API driver function (i.e. IPCLiteXtoYDataWrite()). If the IPC interrupt flag is still busy processing another command, this function will return STATUS_FAIL and not send any command.
2. Processor X's IPC API Driver Function inserts values into XTOYIPCCOM, XTOYIPCADDR, XTOYIPCDATAW, XTOYIPCDATAR registers as needed by IPC command.
3. Processor X's IPC API function then sets XtoY IPC INT flag to trigger an interrupt on processor Y.
4. Processor Y's XtoY IPC INT ISR is triggered. In the ISR, processor Y's application code reads the XTOYIPCCOM register and decodes the command to determine which IPC API driver function to call (i.e. IPCLiteXtoYDataWrite()).
5. The IPC API driver function processes the command, writes any data to return to processor X in the XTOYDATAR register, acknowledges the IPC interrupt flag, and the interrupt ISR is exited.
6. When processor X's application code needs the return data from processor Y for that function, it calls the IPCLiteXtoYGetResult() function in a loop and waits until the function returns STATUS_PASS. At this point, the value sent from processor Y can be read by processor X.
7. Only now can Processor X send another command to Processor Y.

There is only one step the application project must take in order to use the main IPC API driver functions.

- Add the F2837xD_lpc_Driver_Lite.c and F2837xD_lpc_Driver_Util.c files which include the F2837xD_lpc_drivers.h file to the application project. Make sure /common/source/F2837xD_lpc_Driver_Lite.c and /common/source/F2837xD_lpc_Driver_Util.c are added as source files in your project, both of which include /common/include/F2837xD_lpc_drivers.h. The F2837xD_lpc_Driver_Util.c file includes utility functions that can be used by application code and that are used by both the main IPC driver functions and the IPC-lite driver functions.

Both the CPU1 and CPU2 application codes can immediately start calling the IPC command functions. Generally when an IPC command function is called for the sending processor, the

responding function for the receiving processor has the same name. For instance, CPU1 application code might call the [IPCLtoRDataRead\(\)](#) function to write data to memory accessible

only by CPU2. CPU2 will decode the command in the CPU1 TO CPU2 IPCCOM register when the interrupt is flagged, and upon seeing an IPC_DATA_READ command, it will call its own

[IPCRtoLDataRead\(\)](#) function to process the command.

For examples demonstrating many of the basic IPC-Lite functions, see the IPC-Lite examples in the C2000Ware device_support package for your device.

2.2.3 Function Documentation

2.2.3.1 IPCLiteLtoRClearBits

Sets the designated bits in a 16/32-bit data word at the remote CPU system address

Prototype:

```
uint16_t
IPCLiteLtoRClearBits(uint32_t ulFlag,
                     uint32_t ulAddress,
                     uint32_t ulMask,
                     uint16_t usLength,
                     uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Local to Remote IPC Flag number mask used to indicate a command is being sent.

ulAddress specifies the Remote CPU address to write to.

ulMask specifies the 16/32-bit mask for bits which should be set at the remote CPU ulAddress. (For 16-bit mask, only the lower 16-bits of ulMask are considered.

usLength specifies the length of the *ulMask* (1 = 16-bit, 2 = 32-bit).

ulStatusFlag indicates the Local to Remote Flag number mask used to report the status of the command sent back from the Master system.

Description:

This function will allow the Local CPU system to set bits specified by the *usMask* variable in a 16/32-bit word on the Remote CPU system. The data word at /e ulAddress after the set bits command is then read into the IPCREMOTEREPLY register. After calling this function, a call to [IPCLiteLtoRGetResult\(\)](#) will read the data value in the IPCREMOTEREPLY register into a 16/32-bit variable in the Local CPU application. The *usLength* parameter accepts the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The *ulStatusFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**. The function returns **STATUS_PASS** if the command is successful or **STATUS_FAIL** if the request or status flags are unavailable.

Returns:

status of command (0=success, 1=error)

2.2.3.2 IPCLiteLtoRClearBits_Protected

Clears the designated bits in a 16/32-bit write-protected data word at Remote CPU system address

Prototype:

```
uint16_t
IPCLiteLtoRClearBits_Protected(uint32_t ulFlag,
                                uint32_t ulAddress,
                                uint32_t ulMask,
                                uint16_t usLength,
                                uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Local to Remote IPC Flag number mask used to indicate a command is being sent.

ulAddress specifies the Remote CPU write-protected address to write to.

ulMask specifies the 16/32-bit mask for bits which should be cleared at Remote CPU ulAddress. For 16-bit mask, only the lower 16-bits of ulMask are considered.

usLength specifies the length of the **ulMask** (1 = 16-bit, 2 = 32-bit).

ulStatusFlag indicates the Local to Remote Flag number mask used to report the status of the command sent back from the Master system.

Description:

This function will allow the Local CPU system to clear bits specified by the **usMask** variable in a write-protected 16/32-bit word on the Remote CPU system. The data word at /e ulAddress after the clear bits command is then read into the IPCREMOTEREPLY register. After calling this function, a call to [IPCLiteLtoRGetResult\(\)](#) will read the data value in the IPCREMOTEREPLY register into a 16/32-bit variable in the Local CPU application. The **usLength** parameter accepts the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The **ulStatusFlag** parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**. The function returns **STATUS_PASS** if the command is successful or **STATUS_FAIL** if the request or status flags are unavailable.

Returns:

status of command (0=success, 1=error)

2.2.3.3 IPCLiteLtoRDataRead

Reads either a 16- or 32-bit data word from the remote CPU System address

Prototype:

```
uint16_t
IPCLiteLtoRDataRead(uint32_t ulFlag,
                    uint32_t ulAddress,
                    uint16_t usLength,
                    uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Local to Remote IPC Flag number mask used to indicate a command is being sent.

ulAddress specifies the remote address to read from

usLength designates 16- or 32-bit read (1 = 16-bit, 2 = 32-bit)

ulStatusFlag indicates the Local to Remote Flag number mask used to report the status of the command sent back from the remote system.

Description:

This function will allow the Local CPU System to read 16/32-bit data from the Remote CPU System into the IPCREMOTEREPLY register. After calling this function, a call to [IPCLiteLtoRGetResult\(\)](#) will read the data value in the IPCREMOTEREPLY register into a 16- or 32-bit variable in the local CPU application. The *usLength* parameter accepts the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The *ulStatusFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**. The function returns **STATUS_PASS** if the command is successful or **STATUS_FAIL** if the request or status flags are unavailable.

Returns:

status of command (0=success, 1=error)

2.2.3.4 IPCLiteLtoRDataWrite

Writes a 16/32-bit data word to Remote CPU System address

Prototype:

```
uint16_t
IPCLiteLtoRDataWrite(uint32_t ulFlag,
                     uint32_t ulAddress,
                     uint32_t ulData,
                     uint16_t usLength,
                     uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Local to Remote IPC Flag number mask used to indicate a command is being sent.

ulAddress specifies the Remote CPU address to write to

ulData specifies the 16/32-bit word which will be written. For 16-bit words, only the lower 16-bits of ulData will be considered by the master system.

usLength is the length of the word to write (0 = 16-bits, 1 = 32-bits)

ulStatusFlag indicates the Local to Remote Flag number mask used to report the status of the command sent back from the Remote CPU system.

Description:

This function will allow the Local CPU System to write a 16/32-bit word via the *ulData* variable to an address on the Remote CPU System. The *usLength* parameter accepts the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The *ulStatusFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**. The function returns **STATUS_PASS** if the command is successful or **STATUS_FAIL** if the request or status flags are unavailable.

Returns:

status of command (0=success, 1=error)

2.2.3.5 IPCLiteLtoRDataWrite_Protected

Writes a 16/32-bit data word to a protected Remote CPU System address

Prototype:

```
uint16_t
IPCLiteLtoRDataWrite_Protected(uint32_t ulFlag,
                                uint32_t ulAddress,
                                uint32_t ulData,
                                uint16_t usLength,
                                uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Local to Remote IPC Flag number mask used to indicate a command is being sent.

ulAddress specifies the Remote CPU address to write to

ulData specifies the 16/32-bit word which will be written. For 16-bit words, only the lower 16-bits of ulData will be considered by the master system.

usLength is the length of the word to write (0 = 16-bits, 1 = 32-bits)

ulStatusFlag indicates the Local to Remote Flag number mask used to report the status of the command sent back from the Master system.

Description:

This function will allow the Local CPU System to write a 16/32-bit word via the *ulData* variable to a write-protected address on the Remote CPU System. The *usLength* parameter accepts the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The *ulStatusFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**. The function returns **STATUS_PASS** if the command is successful or **STATUS_FAIL** if the request or status flags are unavailable.

Returns:

status of command (0=success, 1=error)

2.2.3.6 IPCLiteLtoRFunctionCall

Calls a Remote CPU function with 1 optional parameter and an optional return value.

Prototype:

```
uint16_t
IPCLiteLtoRFunctionCall(uint32_t ulFlag,
                        uint32_t ulAddress,
                        uint32_t ulParam,
                        uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Local to Remote IPC Flag number mask used to indicate a command is being sent.

ulAddress specifies the Remote CPU function address

ulParam specifies the 32-bit optional parameter value

ulStatusFlag indicates the Local to Remote Flag number mask used to report the status of the command sent back from the control system.

Description:

This function will allow the Local CPU system to call a function on the Remote CPU. The *ulParam* variable is a single optional 32-bit parameter to pass to the function. The *ulFlag* param-

eter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32**. The *ulStatusFlag* parameter accepts any other one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**. The function returns **STATUS_PASS** if the command is successful or **STATUS_FAIL** if the request or status flags are unavailable.

Returns:

status of command (0=success, 1=error)

2.2.3.7 IPCLiteLtoRGetResult

Reads single word data result of Local to Remote IPC command

Prototype:

```
uint16_t
IPCLiteLtoRGetResult(void *pvData,
                    uint16_t usLength,
                    uint32_t ulStatusFlag)
```

Parameters:

pvData is a pointer to the 16/32-bit variable where the result data will be stored.

usLength designates 16- or 32-bit read.

ulStatusFlag indicates the Local to Remote CPU Flag number mask used to report the status of the command sent back from the Remote CPU. If a status flag was not used with the command call, set this parameter to 0.

Description:

Allows the caller to read the 16/32-bit data result of non-blocking IPC functions from the IPCRE-MOTEREPLY register if the status flag is cleared indicating the IPC command was successfully interpreted. If the status flag is not cleared, the command was not recognized, and the function will return **STATUS_FAIL**. To determine what data is read from a call to this function, see the descriptions of the non-blocking IPC functions. The *usLength* parameter accepts the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The *ulStatusFlag* parameter accepts any of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**. The function returns **STATUS_PASS** or **STATUS_FAIL**.

Returns:

status of command (0=success, 1=error)

2.2.3.8 IPCLiteLtoRSetBits

Sets the designated bits in a 16/32-bit data word at the remote CPU system address

Prototype:

```
uint16_t
IPCLiteLtoRSetBits(uint32_t ulFlag,
                  uint32_t ulAddress,
                  uint32_t ulMask,
                  uint16_t usLength,
                  uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Local to Remote IPC Flag number mask used to indicate a command is being sent.

ulAddress specifies the Remote address to write to.

ulMask specifies the 16/32-bit mask for bits which should be set at remote ulAddress. For 16-bit mask, only the lower 16-bits of ulMask are considered.

usLength specifies the length of the *ulMask* (1 = 16-bit, 2 = 32-bit).

ulStatusFlag indicates the Local to Remote Flag number mask used to report the status of the command sent back from the Remote system.

Description:

This function will allow the Local CPU system to set bits specified by the *usMask* variable in a 16/32-bit word on the Remote CPU system. The data word at /e ulAddress after the set bits command is then read into the IPCREMOTEREPLY register. After calling this function, a call to [IPCLiteLtoRGetResult\(\)](#) will read the data value in the IPCREMOTEREPLY register into a 16/32-bit variable in the Local CPU application. The *usLength* parameter accepts the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The *ulStatusFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**. The function returns **STATUS_PASS** if the command is successful or **STATUS_FAIL** if the request or status flags are unavailable.

Returns:

status of command (0=success, 1=error)

2.2.3.9 IPCLiteLtoRSetBits_Protected

Sets the designated bits in a 16/32-bit write-protected data word at the Remote CPU system address

Prototype:

```
uint16_t
IPCLiteLtoRSetBits_Protected(uint32_t ulFlag,
                             uint32_t ulAddress,
                             uint32_t ulMask,
                             uint16_t usLength,
                             uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Local to Remote IPC Flag number mask used to indicate a command is being sent.

ulAddress specifies the Remote CPU write-protected address to write to.

ulMask specifies the 16/32-bit mask for bits which should be set at Remote CPU ulAddress. For 16-bit mask, only the lower 16-bits of ulMask are considered.

usLength specifies the length of the *ulMask* (1 = 16-bit, 2 = 32-bit).

ulStatusFlag indicates the Local to Remote Flag number mask used to report the status of the command sent back from the Master system.

Description:

This function will allow the Local CPU system to set bits specified by the *usMask* variable in a write-protected 16/32-bit word on the Remote CPU system. The data word at /e ulAddress after the set bits command is then read into the IPCREMOTEREPLY register. After calling this

function, a call to [*IPCLiteLtoRGetResult\(\)*](#) will read the data value in the IPCREMOTEREPLY register into a 16/32-bit variable in the Local application. The *usLength* parameter accepts the following values: **IPC_LENGTH_16_BITS** or **IPC_LENGTH_32_BITS**. The *ulStatusFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**. The function returns **STATUS_PASS** if the command is successful or **STATUS_FAIL** if the request or status flags are unavailable.

Returns:

status of command (0=success, 1=error)

2.2.3.10 IPCLiteReqMemAccess

Slave Requests Master R/W/Exe Access to Shared SARAM.

Prototype:

```
uint16_t
IPCLiteReqMemAccess (uint32_t ulFlag,
                    uint32_t ulMask,
                    uint16_t ulMaster,
                    uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Local to Remote IPC Flag number mask used to indicate a command is being sent.

ulMask specifies the 32-bit mask for the GSxMEMSEL RAM control register to indicate which GSx SARAM blocks the Slave is requesting master access to.

ulMaster specifies whether CPU1 or CPU2 should be the master of the GSx RAM.

ulStatusFlag indicates the Local to Remote Flag number mask used to report the status of the command sent back from the Master system.

Description:

This function will allow the slave CPU System to request slave or master mastership of any of the GSx Shared SARAM blocks. The *ulMaster* parameter accepts the following values: **IPC_GSX_CPU2_MASTER** or **IPC_GSX_CPU1_MASTER**. The *ulStatusFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**. The function returns **STATUS_PASS** if the command is successful or **STATUS_FAIL** if the request or status flags are unavailable.

Note:

This function calls the [*IPCLiteLtoRSetBits_Protected\(\)*](#) or the [*IPCLiteLtoRClearBits_Protected*](#) function, and therefore in order to process this function, the above 2 functions should be ready to be called on the master system to process this command.

Returns:

status of command (0=success, 1=error)

2.2.3.11 IPCLiteRtoLClearBits

Clears the designated bits in a 16/32-bit data word at Local CPU system address

Prototype:

```
void
IPCLiteRtoLClearBits(uint32_t ulFlag,
                     uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Remote to Local IPC Flag number mask used to indicate a command is being sent.

ulStatusFlag indicates the Remote to Local Flag number mask used to report the status of the command sent back from the control system.

Description:

This function will allow the Remote CPU system to clear bits specified by a mask variable in a 16/32-bit word on the Local CPU system, and then read back the word into the IPCLOCALREPLY register. The *ulFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32**, and the *ulStatusFlag* parameter accepts any other one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**.

2.2.3.12 void IPCLiteRtoLClearBits_Protected (uint32_t *ulFlag*, uint32_t *ulStatusFlag*)

Clears the designated bits in a 16/32-bit data word at the Local CPU system write-protected address

Parameters:

ulFlag specifies Remote to Local IPC Flag number mask used to indicate a command is being sent.

ulStatusFlag indicates the Remote to Local Flag number mask used to report the status of the command sent back from the control system.

This function will allow the Remote CPU system to clear bits specified by a mask variable in a 16/32-bit word on the Local CPU system, and then read back the word into the IPCLOCALREPLY register. The *ulFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32**, and the *ulStatusFlag* parameter accepts any other one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**.

2.2.3.13 IPCLiteRtoLDataRead

Reads either a 16- or 32-bit data word from the Local CPU system address

Prototype:

```
void
IPCLiteRtoLDataRead(uint32_t ulFlag,
                    uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Remote to Local IPC Flag number mask used to indicate a command is being sent.

ulStatusFlag indicates the Remote to Local Flag number mask used to report the status of the command sent back from the control system.

Description:

This function will allow the Remote CPU system to read 16/32-bit data from the Local CPU system. The *ulFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32**, and the *ulStatusFlag* parameter accepts any other one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**.

2.2.3.14 void IPCLiteRtoLDataWrite (uint32_t ulFlag, uint32_t ulStatusFlag)

Writes a 16/32-bit data word to Local CPU system address

Parameters:

ulFlag specifies Remote to Local IPC Flag number mask used to indicate a command is being sent.

ulStatusFlag indicates the Remote to Local Flag number mask used to report the status of the command sent back from the control system.

This function will allow the Remote CPU system to write a 16/32-bit word to an address on the Local CPU system. The *ulFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32**, and the *ulStatusFlag* parameter accepts any other one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**.

2.2.3.15 IPCLiteRtoLDataWrite_Protected

Writes a 16/32-bit data word to a write-protected Local CPU system address

Prototype:

```
void  
IPCLiteRtoLDataWrite_Protected(uint32_t ulFlag,  
                                uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Remote to Local IPC Flag number mask used to indicate a command is being sent.

ulStatusFlag indicates the Remote to Local Flag number mask used to report the status of the command sent back from the control system.

Description:

This function will allow the Remote CPU system to write a 16/32-bit word to an address on the Local CPU system. The *ulFlag* parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32**, and the *ulStatusFlag* parameter accepts any other one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**.

2.2.3.16 void IPCLiteRtoLFunctionCall (uint32_t ulFlag, uint32_t ulStatusFlag)

Calls a Local CPU function with a single optional parameter and return value.

Parameters:

ulFlag specifies Remote to Local IPC Flag number mask used to indicate a command is being sent.

ulStatusFlag indicates the Remote to Local Flag number mask used to report the status of the command sent back from the control system.

This function will allow the Remote CPU system to call a Local CPU function with a single optional parameter and places an optional return value in the IPCLOCALREPLY register. The ***ulFlag*** parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32**, and the ***ulStatusFlag*** parameter accepts any other one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**.

2.2.3.17 IPCLiteRtoLSetBits

Sets the designated bits in a 16/32-bit data word at the Local CPU system address

Prototype:

```
void
IPCLiteRtoLSetBits(uint32_t ulFlag,
                   uint32_t ulStatusFlag)
```

Parameters:

ulFlag specifies Remote to Local IPC Flag number mask used to indicate a command is being sent.

ulStatusFlag indicates the Remote to Local Flag number mask used to report the status of the command sent back from the control system.

Description:

This function will allow the Remote CPU system to set bits specified by a mask variable in a 16/32-bit word on the Local CPU system, and then read back the word into the IPCLOCALREPLY register. The ***ulFlag*** parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32**, and the ***ulStatusFlag*** parameter accepts any other one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**.

2.2.3.18 void IPCLiteRtoLSetBits_Protected (uint32_t ulFlag, uint32_t ulStatusFlag)

Sets the designated bits in a 16-bit data word at the Local CPU system write-protected address

Parameters:

ulFlag specifies Remote to Local IPC Flag number mask used to indicate a command is being sent.

ulStatusFlag indicates the Remote to Local Flag number mask used to report the status of the command sent back from the control system.

This function will allow the Remote CPU system to set bits specified by a mask variable in a write-protected 16/32-bit word on the Local CPU system, and then read back the word into the IPCLOCALREPLY register. The ***ulFlag*** parameter accepts any one of the flag values **IPC_FLAG1** - **IPC_FLAG32**, and the ***ulStatusFlag*** parameter accepts any other one of the flag values **IPC_FLAG1** - **IPC_FLAG32** and **NO_FLAG**.

2.3 IPC Utility Drivers

Functions

- uint32_t [IPCGetBootStatus](#) (void)
- Uint16 [IPCLtoRFlagBusy](#) (uint32_t ulFlags)
- void [IPCLtoRFlagClear](#) (uint32_t ulFlags)
- void [IPCLtoRFlagSet](#) (uint32_t ulFlags)
- void [IPCRtoLFlagAcknowledge](#) (uint32_t ulFlags)
- Uint16 [IPCRtoLFlagBusy](#) (uint32_t ulFlags)

2.3.1 Detailed Description

The IPC Utility driver functions provide convenient functions to set/clear/check the status of IPC flags and a function to allow CPU1 to run the CPU2 peripheral loaders while the C28 is in boot mode. These functions can be used in conjunction with either the IPC-Lite or main IPC drivers as long as the F2837xD_Ipc_Driver_Util.c file is added to the project.

2.3.2 Function Documentation

2.3.2.1 IPCGetBootStatus

Local Return CPU02 BOOT status

Prototype:

```
uint32_t
IPCGetBootStatus(void)
```

Description:

This function returns the value at IPCBOOTSTS register.

Returns:

Boot status.

2.3.2.2 IPCLtoRFlagBusy

Determines whether the given IPC flags are busy or not.

Prototype:

```
Uint16
IPCLtoRFlagBusy(uint32_t ulFlags)
```

Parameters:

ulFlags specifies Local to Remote IPC Flag number masks to check the status of.

Description:

Allows the caller to determine whether the designated IPC flags are available for further control to master system communication. If **0** is returned, then all designated tasks have completed and are available. The *ulFlags* parameter can be any of the IPC flag values: **IPC_FLAG0** - **IPC_FLAG31**.

Returns:

Returns **1** if the IPC flags are busy or **0** if designated IPC flags are free.

2.3.2.3 IPCLtoRFlagClear

Local CPU Clears Local to Remote IPC Flag

Prototype:

```
void  
IPCLtoRFlagClear(uint32_t ulFlags)
```

Parameters:

ulFlags specifies the IPC flag mask for flags being set.

Description:

This function will allow the Local CPU system to set the designated IPC flags to send to the Remote CPU system. The *ulFlags* parameter can be any of the IPC flag values: **IPC_FLAG0** - **IPC_FLAG31**.

Returns:

None.

2.3.2.4 IPCLtoRFlagSet

Local CPU Sets Local to Remote IPC Flag

Prototype:

```
void  
IPCLtoRFlagSet(uint32_t ulFlags)
```

Parameters:

ulFlags specifies the IPC flag mask for flags being set.

Description:

This function will allow the Local CPU system to set the designated IPC flags to send to the Remote CPU system. The *ulFlags* parameter can be any of the IPC flag values: **IPC_FLAG0** - **IPC_FLAG31**.

Returns:

None.

2.3.2.5 IPCRtoLFlagAcknowledge

Local CPU Acknowledges Remote to Local IPC Flag.

Prototype:

```
void  
IPCRtoLFlagAcknowledge(uint32_t ulFlags)
```

Parameters:

ulFlags specifies the IPC flag mask for flags being acknowledged.

Description:

This function will allow the Local CPU system to acknowledge/clear the IPC flag set by the Remote CPU system. The *ulFlags* parameter can be any of the IPC flag values: **IPC_FLAG0** - **IPC_FLAG31**.

Returns:

None.

2.3.2.6 IPCRtoLFlagBusy

Determines whether the given Remote to Local IPC flags are busy or not.

Prototype:

```
Uint16  
IPCRtoLFlagBusy(uint32_t ulFlags)
```

Parameters:

ulFlags specifies Remote to Local IPC Flag number masks to check the status of.

Description:

Allows the caller to determine whether the designated IPC flags are pending. The *ulFlags* parameter can be any of the IPC flag values: **IPC_FLAG0** - **IPC_FLAG31**.

Returns:

Returns **1** if the IPC flags are busy or **0** if designated IPC flags are free.

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

Amplifiers	amplifier.ti.com
Data Converters	dataconverter.ti.com
DLP® Products	www.dlp.com
DSP	dsp.ti.com
Clocks and Timers	www.ti.com/clocks
Interface	interface.ti.com
Logic	logic.ti.com
Power Mgmt	power.ti.com
Microcontrollers	microcontroller.ti.com
RFID	www.ti-rfid.com
RF/IF and ZigBee® Solutions	www.ti.com/lprf

Applications

Audio	www.ti.com/audio
Automotive	www.ti.com/automotive
Broadband	www.ti.com/broadband
Digital Control	www.ti.com/digitalcontrol
Medical	www.ti.com/medical
Military	www.ti.com/military
Optical Networking	www.ti.com/opticalnetwork
Security	www.ti.com/security
Telephony	www.ti.com/telephony
Video & Imaging	www.ti.com/video
Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2019, Texas Instruments Incorporated