PrimeCell® Inter-Processor Communications Module (PL320)

Revision: r0p0

Technical Reference Manual

ARM DDI 0306B

PrimeCell Inter-Processor Communications Module (PL320)
Technical Reference Manual

Release Information

The following changes have been made to this document.

Change history

Date	Issue	Change
19 December 2003	A	First release
22 June 2004	B	Reclassify to open access for r0p0

Proprietary Notice

Words and logos marked with ® or ™ are registered trademarks or trademarks of ARM Limited in the EU and other countries, except as otherwise stated below in this proprietary notice. Other brands and names mentioned herein may be the trademarks of their respective owners.

Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder.

The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given by ARM Limited in good faith. However, all warranties implied or expressed, including but not limited to implied warranties of merchantability, or fitness for purpose, are excluded.

This document is intended only to assist the reader in the use of the product. ARM Limited shall not be liable for any loss or damage arising from the use of any information in this document, or any error or omission in such information, or any incorrect use of the product.

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by ARM and the party that ARM delivered this document to.

Product Status

The information in this document is final, that is for a developed product.

Web Address

`http://www.arm.com`

Contents
PrimeCell Inter-Processor Communications Module (PL320) Technical Reference Manual

Preface

Chapter 1 Introduction

Chapter 2 Functional Overview

Chapter 3 Programmer’s Model

Chapter 4 Programmer’s Model for Test

List of Tables
PrimeCell Inter-Processor Communications Module (PL320) Technical Reference Manual

List of Figures
PrimeCell Inter-Processor Communications Module (PL320) Technical Reference Manual

Preface

This preface introduces the *PrimeCell Inter-Processor Communications Module Revision r0p0 PrimeCell Inter-Processor Communications Module (PL320) Technical Reference Manual* (TRM). It contains the following sections:

- *About this manual* on page x
- *Feedback* on page xiv.

About this manual

This is the TRM for the *Inter-Processor Communications Module* (IPCM).

Product revision status

The r*n*p*n* identifier indicates the revision status of the product described in this manual, where:

r*n* Identifies the major revision of the product.

p*n* Identifies the minor revision or modification status of the product.

Intended audience

This manual is written for hardware engineers who have some experience of using ARM SoC design flow and methodology. Prior experience of the PrimeCell IPCM is not assumed.

Using this manual

This manual is organized into the following chapters:

Chapter 1 *Introduction*

Read this chapter for an introduction to the IPCM and its features.

Chapter 2 *Functional Overview*

Read this chapter for a description of the major functional blocks of the IPCM.

Chapter 3 *Programmer's Model*

Read this chapter for a description of the IPCM registers and programming details.

Chapter 4 *Programmer's Model for Test*

Read this chapter for a description of the logic in the IPCM for functional verification and production testing.

Appendix A *Signal Descriptions*

Read this appendix for details of the IPCM signals.

Conventions

Conventions that this manual can use are described in:

- *Typographical*
- *Timing diagrams*
- *Signals* on page xii
- *Numbering* on page xiii.

Typographical

The typographical conventions are:

italic Highlights important notes, introduces special terminology, denotes internal cross-references, and citations.

bold Highlights interface elements, such as menu names. Denotes signal names. Also used for terms in descriptive lists, where appropriate.

`monospace` Denotes text that you can enter at the keyboard, such as commands, file and program names, and source code.

`monospace` Denotes a permitted abbreviation for a command or option. You can enter the underlined text instead of the full command or option name.

`monospace italic` Denotes arguments to monospace text where the argument is to be replaced by a specific value.

`monospace bold` Denotes language keywords when used outside example code.

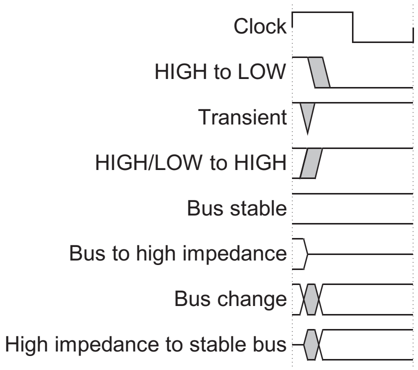
< and > Angle brackets enclose replaceable terms for assembler syntax where they appear in code or code fragments. They appear in normal font in running text. For example:

- `MRC p15, 0 <Rd>, <CRn>, <CRm>, <Opcode_2>`
- The Opcode_2 value selects which register is accessed.

Timing diagrams

The figure named *Key to timing diagram conventions* on page xii explains the components used in timing diagrams. Variations, when they occur, have clear labels. You must not assume any timing information that is not explicit in the diagrams.

Shaded bus and signal areas are undefined, so the bus or signal can assume any value within the shaded area at that time. The actual level is unimportant and does not affect normal operation.



Key to timing diagram conventions

Signals

The signal conventions are:

Signal level The level of an asserted signal depends on whether the signal is active-HIGH or active-LOW. Asserted means HIGH for active-HIGH signals and LOW for active-LOW signals.

Prefix A Denotes *Advanced eXtensible Interface* (AXI) global and address channel signals.

Prefix B Denotes AXI write response channel signals.

Prefix C Denotes AXI low-power interface signals.

Prefix H Denotes *Advanced High-performance Bus* (AHB) signals.

Prefix n Denotes active-LOW signals except in the case of AXI, AHB or *Advanced Peripheral Bus* (APB) reset signals.

Prefix P Denotes APB signals.

Prefix R Denotes AXI read channel signals.

Prefix W Denotes AXI write channel signals.

Suffix n Denotes AXI, AHB, and APB reset signals.

Numbering

The numbering convention is:

<size in bits>'<base><number>

This is a Verilog method of abbreviating constant numbers. For example:

- 'h7B4 is an unsized hexadecimal value.
- 'o7654 is an unsized octal value.
- 8'd9 is an eight-bit wide decimal value of 9.
- 8'h3F is an eight-bit wide hexadecimal value of `0x3F`. This is equivalent to b00111111.
- 8'b1111 is an eight-bit wide binary value of b00001111.

Further reading

This section lists publications by ARM Limited, and by third parties.

ARM Limited periodically provides updates and corrections to its documentation. See `http://www.arm.com` for current errata sheets, addenda, and the ARM Limited Frequently Asked Questions list.

ARM publications

This manual contains information that is specific to the IPCM. Refer to the following documents for other relevant information:

- *AMBA® Specification (Rev 2.0)* (ARM IHI 0011)
- *AMBA AXI Protocol Specification* (ARM IHI 0022)
- *DSP Integration Specification* (ARM IHI 0026)
- *Message Passing Software Integration Guide* (ARM DII 0091)
- *PrimeCell Inter-Processor Communications Module Implementation Guide* (ARM DII 0107).
- *PrimeCell Inter-Processor Communications Module Integration Manual* (ARM DII 0108).
- *PrimeCell Vectored Interrupt Controller (PL192) Technical Reference Manual* (ARM DDI 0273)
- *PrimeCell Core Identification Module (PL321) r0p0 Technical Reference Manual* (ARM DDI 0327).

Feedback

ARM Limited welcomes feedback on the IPCM and its documentation.

Feedback on the IPCM

If you have any comments or suggestions about this product, contact your supplier giving:

- the product name
- a concise explanation of your comments.

Feedback on this manual

If you have any comments on this manual, send email to `errata@arm.com` giving:

- the title
- the number
- the relevant page number(s) to which your comments apply
- a concise explanation of your comments.

ARM Limited also welcomes general suggestions for additions and improvements.

Chapter 1
Introduction

This chapter introduces the *Inter-Processor Communications Module* (IPCM). It contains the following section:

- *About the IPCM* on page 1-2.

1.1 About the IPCM

The IPCM provides up to 32 mailboxes with control logic and interrupt generation to support inter-processor communication. An AHB interface enables access from source and destination cores.

The IPCM:

- sends interrupts to other cores
- passes small amounts of data to other cores.

The mailboxes within the IPCM can be available as floating resources between cores or as dedicated resources to specific cores. A source core can have multiple mailboxes and send messages in parallel.

The IPCM consists of the following:

- 1-32 programmable mailboxes, each comprising:
  - a single 1-32-bit Mailbox Source Register
  - a single 1-32-bit Mailbox Destination Register with separate Set, Clear, and Status addresses
  - a single 2-bit Mailbox Mode Register to enable Auto Acknowledge and Auto Link modes
  - a single 1-32-bit Mailbox Mask Register with separate Set, Clear, and Status addresses to enable you to mask out individual mailbox interrupts for cores requiring to poll rather than be interrupted
  - a single 2-bit Mailbox Send Register to trigger mailbox interrupts to source and destination cores
  - 0-7 32-bit data registers to store the message.
- 1-32 sets of read-only interrupt status registers, one for each interrupt, each comprising:
  - 1-32-bit Raw Interrupt Status Register (each bit corresponds to each mailbox)
  - 1-32-bit Masked Interrupt Status Register (each bit corresponds to each mailbox).
- A 32-bit Configuration Status Register
- Integration Test Registers for the interrupt outputs
- Peripheral and PrimeCell Identification Registers.

The IPCM is a highly configurable and programmable module. It has three configurable parameters:

- 1-32 mailboxes
- 0-7 data registers per mailbox
- 1-32 interrupts.

These parameters reduce gate count by enabling you to configure the IPCM instance to match the system requirements. The programmable features, such as source, destination, mode, and mask, enable the configured IPCM to be used by different cores in different ways, depending on the current application.

Chapter 2
Functional Overview

This chapter describes the major functional blocks of the IPCM. It contains the following sections:

- *Functional description* on page 2-2
- *Functional operation* on page 2-4
- *Examples of messaging* on page 2-18.

2.1 Functional description

Figure 2-1 shows a block diagram of the IPCM.

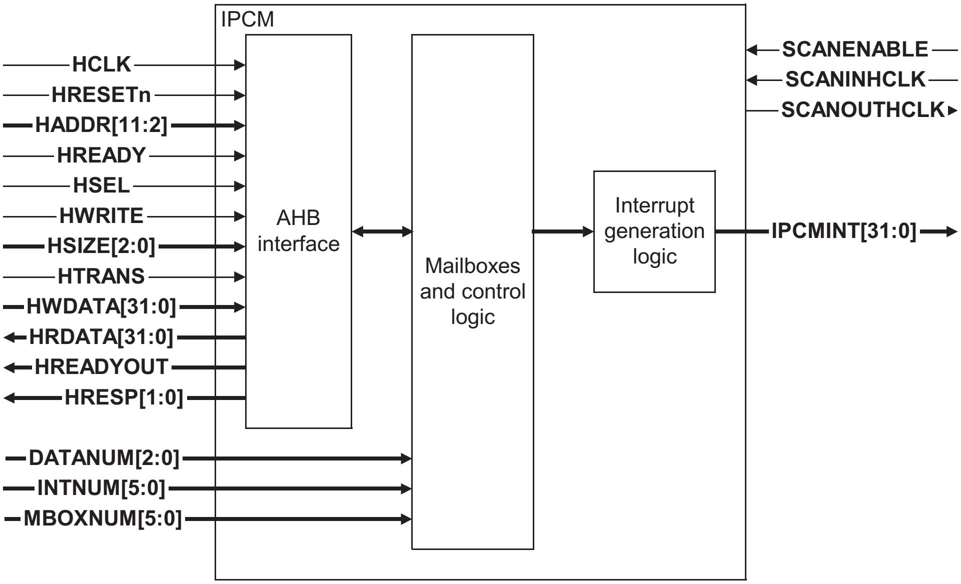


Figure 2-1 IPCM block diagram

The IPCM contains three main functional blocks:

AHB interface

The AHB interface enables access from the system bus to the IPCM registers.

Mailboxes and control logic

The mailbox and control logic block contains all the mailbox registers and control logic.

Interrupt generation logic

The interrupt generation logic block generates the IPCM interrupt outputs from the current status of all the IPCM mailboxes.

Figure 2-2 on page 2-3 shows the integration of the IPCM in a multiprocessing system.

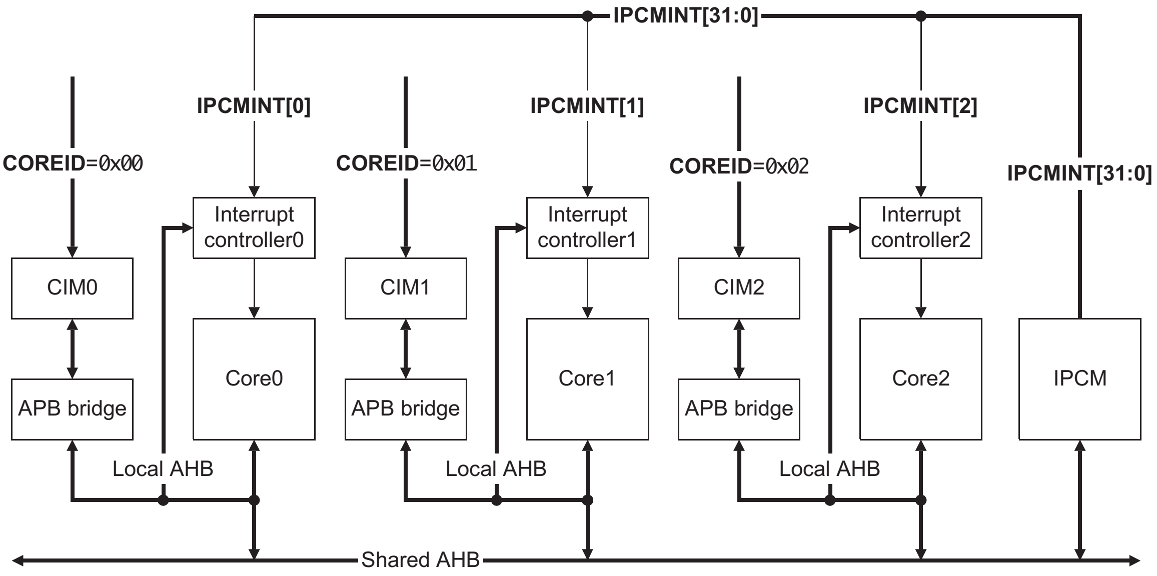


Figure 2-2 IPCM integration in a multiprocessing system

For information on the *Core Identification Module* (CIM), see the *ARM PrimeCell Core Identification Module (PL321) r0p0 Technical Reference Manual*.

2.2 Functional operation

The IPCM generates interrupts under software control. These interrupts normally have data associated with them and can be directed to one or more of up to 32 different interrupt outputs. Each interrupt output corresponds directly to a bit in every Mailbox Source, Mailbox Destination, and Mailbox Mask Register in every mailbox in the IPCM. These registers therefore control which interrupt lines are asserted when messages are sent and acknowledged.

You connect the interrupt outputs to the system interrupt controllers during integration. One or more interrupt outputs can connect to each interrupt controller. Normally the IPCM has at least one interrupt output connected to every interrupt controller in the system, enabling any core to send a message to any other core. When more than one IPCM interrupt is connected to the same interrupt controller, different types of message can be indicated on different interrupt lines, and therefore handled by different ISRs.

A multi-core system normally has at least one IPCM instantiated. More can be instantiated if required. Because the IPCM is configurable, you can have several differently configured IPCMs instantiated in the same system.

The operation of the IPCM is described in more detail in the following sections:

- *Basic operation*
- *Channel ID* on page 2-5
- *Using mailboxes* on page 2-7.

2.2.1 Basic operation

Figure 2-3 on page 2-5 shows an example system in which the IPCM is integrated so that **IPCMINT[0]** is connected to the interrupt controller for Core0 and **IPCMINT[1]** is connected to the interrupt controller for Core1.

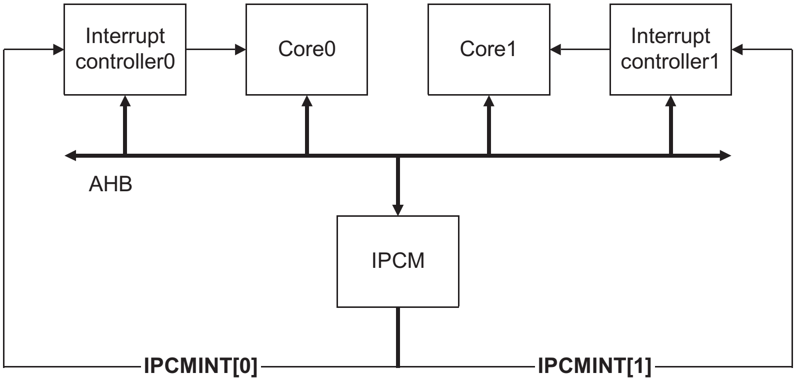


Figure 2-3 Basic operation

The IPCM operates as follows:

1. Core0 has a message to send to Core1. Core0 claims the mailbox by setting bit 0 in the Mailbox Source Register. Core0 then sets bit 1 in the Mailbox Destination Register, enables the interrupts and programs the message into the Mailbox Data Registers. Finally, Core0 sends the message by writing 01 to the Mailbox Send Register. This asserts the interrupt to Core1.

2. When Core1 is interrupted, it reads the Masked Interrupt Status Register for **IPCMINT[1]** to determine which mailbox contains the message. Core1 reads the message in that mailbox, then clears the interrupt and asserts the acknowledge interrupt by writing 10 to the Mailbox Send Register.

3. Core0 is interrupted with the acknowledge message, completing the operation. Core0 then decides whether to retain the mailbox to send another message or release the mailbox, freeing it up for other cores in the system to use it.

2.2.2 Channel ID

The Channel ID is defined as the one-hot encoded value that corresponds to a specific interrupt output from the IPCM. An IPCM configured to have 32 interrupt outputs has 32 corresponding Channel IDs. The Channel ID programs the Mailbox Source, Mailbox Destination, and Mailbox Mask Registers.

Table 2-1 shows how Channel IDs map to 32 interrupt outputs.

Table 2-1 Channel ID to interrupt mapping

Channel ID	Interrupt output
`0x00000001`	**IPCMINT[0]**
`0x00000002`	**IPCMINT[1]**
`0x00000004`	**IPCMINT[2]**
`0x00000008`	**IPCMINT[3]**
`0x00000010`	**IPCMINT[4]**
`0x00000020`	**IPCMINT[5]**
`0x00000040`	**IPCMINT[6]**
`0x00000040`	**IPCMINT[7]**
`0x00000100`	**IPCMINT[8]**
`0x00000200`	**IPCMINT[9]**
`0x00000400`	**IPCMINT[10]**
`0x00000800`	**IPCMINT[11]**
`0x00001000`	**IPCMINT[12]**
`0x00002000`	**IPCMINT[13]**
`0x00004000`	**IPCMINT[14]**
`0x00008000`	**IPCMINT[15]**
`0x00010000`	**IPCMINT[16]**
`0x00020000`	**IPCMINT[17]**
`0x00040000`	**IPCMINT[18]**
`0x00080000`	**IPCMINT[19]**
`0x00100000`	**IPCMINT[20]**
`0x00200000`	**IPCMINT[21]**
`0x00400000`	**IPCMINT[22]**
`0x00800000`	**IPCMINT[23]**

Table 2-1 Channel ID to interrupt mapping (continued)

Channel ID	Interrupt output
`0x01000000`	**IPCMINT[24]**
`0x02000000`	**IPCMINT[25]**
`0x04000000`	**IPCMINT[26]**
`0x08000000`	**IPCMINT[27]**
`0x10000000`	**IPCMINT[28]**
`0x20000000`	**IPCMINT[29]**
`0x40000000`	**IPCMINT[30]**
`0x80000000`	**IPCMINT[31]**

Note

The configured number of interrupt outputs defines the width of the Channel ID.

In a system that has one IPCM interrupt per core, each core has a single Channel ID that defines it within the IPCM. Some systems can have multiple IPCM interrupts per core, and therefore multiple Channel IDs per core.

2.2.3 Using mailboxes

This section describes:

- *Defining source core* on page 2-8
- *Defining destination core* on page 2-8
- *Using the Mailbox Mask Register* on page 2-8
- *Using the Mailbox Send Register* on page 2-9
- *Mailbox Data Registers* on page 2-9
- *Setting mode* on page 2-9
- *Interrupts and status Registers* on page 2-10
- *Configuration Status Register* on page 2-12
- *Usage constraints* on page 2-16.

Defining source core

A core must obtain a mailbox to send a message. To do this the core writes one of its Channel IDs to the Mailbox Source Register and then reads the Mailbox Source Register back again to check whether the write was successful. The Mailbox Source Register must only contain a one-hot encoded value, that is, a single Channel ID. The software must ensure that only a one-hot encoded number is written to the Mailbox Source Register. You can only clear the Mailbox Source Register after it is programmed. Any writes other than `0x00000000` are ignored. This mechanism guarantees that only a single core has control of the mailbox at any one time.

A core gives up a mailbox, when it is no longer required, by clearing the Mailbox Source Register. Clearing the Mailbox Source Register also clears all the other registers in the mailbox. This guarantees that a mailbox is always cleared when it is newly allocated.

Defining destination core

The Mailbox Destination Register has separate Set and Clear write locations to enable you to set individual bits in the Mailbox Destination Register without using read-modify-write transfers. You can set a single bit in the Mailbox Destination Register by writing that bit to the Destination Set Register. This causes the hardware to OR that bit with the current Mailbox Destination Register value. Similarly, you can clear a single bit in the Mailbox Destination Register by writing that bit to the Destination Clear Register.

When the source core defines the mode of a mailbox, it defines which other cores are to receive the message by programming the OR of all the Channel IDs into the Mailbox Destination Register. If a core has more than one Channel ID only one is used per message. You can only write to the Mailbox Destination Register after the Mailbox Source Register is defined.

Using the Mailbox Mask Register

The Mailbox Mask Register uses separate Set and Clear registers for modification similar to the Mailbox Destination Register. The Mailbox Mask Register enables the interrupt outputs. To enable interrupts for a particular mailbox, a core writes its Channel ID to the Mask Set location. The interrupt for that mailbox can be masked out by writing the same Channel ID to the Mask Clear location. You can only write to the Mailbox Mask Register locations after the Mailbox Source Register is defined.

Using the Mailbox Send Register

A message is sent by setting bit 0 of the Mailbox Send Register. This triggers the interrupt to the destination core. Clearing this bit clears the interrupt to the destination core. The acknowledge message is sent to the source core by setting bit 1 of the Mailbox Send Register. Clearing this bit clears the interrupt to the source core. You can use one write to clear bit 0 and set bit 1 in the Mailbox Send Register, although this is not mandatory. You cannot set bit 1 then clear bit 0 because 11 is an invalid value for the Mailbox Send Register. The Mailbox Send Register can only be written to after the Mailbox Source Register is defined.

Mailbox Data Registers

The Mailbox Data Registers are general-purpose 32-bit registers that contain the message and can only be written to after the Mailbox Source Register is defined. The Mailbox Data Registers are normally written to before sending the message.

Setting mode

The Mailbox Mode Register controls how the acknowledge interrupt is sent back to the source core, and whether the current mailbox is linked to the next mailbox in the IPCM. The Mailbox Mode Register has two bits and you can only write to it after the Mailbox Source Register is defined.

Auto Acknowledge

In Auto Acknowledge mode, an acknowledge interrupt is automatically sent to the source core after the final destination core has cleared its interrupt. Destination cores must clear their interrupts by writing their Channel ID value to the Destination Clear location. This clears their Channel ID from the Mailbox Destination Register. When the Mailbox Destination Register finally reaches zero, indicating that all destination cores have cleared their interrupts, the mailbox automatically detects this, clears bit 0 and sets bit 1 of the Mailbox Send Register. The source core then receives the acknowledge interrupt. The data associated with an Auto Acknowledge is the same as that for the original message. You can use Auto Acknowledge mode for 1-32 destination cores.

Note

You can use Auto Acknowledge when the system contains just two cores, a source core and a destination core.

When Auto Acknowledge mode is disabled, the acknowledge interrupt is optional. The destination core must clear its interrupt by clearing bit 0 of the Mailbox Send Register. Only when the destination core sets bit 1 of the Mailbox Send Register does the source core obtain its acknowledge interrupt, indicating that the destination core has finished with the message. You can only disable Auto Acknowledge mode when there is only one destination core, where there is also a possibility of updating the message for the acknowledge.

Auto Link

Auto Link provides a mechanism to link mailboxes together so that when a message is acknowledged in one mailbox, the next message is sent from the linked mailbox instead of interrupting the source core. When Auto Link is enabled, the destination core clears bit 0 and sets bit 1 of the Mailbox Send Register in the usual way, but the acknowledge interrupt to the source core is masked out and Mailbox Send Register bit 0 is set in the next mailbox, sending that message.

In this mode, a source core can allocate multiple mailboxes to itself, link them together by setting the Auto Link bits and preload messages in all the mailboxes. When the first message is sent, it is not acknowledged until all the messages have been sent. There is no restriction on the destinations of these messages or whether Auto Acknowledge is enabled when Auto Link is used. In the IPCM, Mailbox0 can be linked to Mailbox1, which in turn can be linked to Mailbox2, up to Mailbox31. For example, if you want to link Mailbox0, Mailbox1, and Mailbox2, set the Auto Link bits in Mailbox0 and Mailbox1. Do not set the Auto Link bit in Mailbox2, to enable the acknowledge interrupt to be sent back to the source core.

When Auto Link is disabled, the source core is interrupted if an acknowledge interrupt is sent that has no effect on any other mailbox.

Note

When using Auto Link with Auto Acknowledge, the mailbox automatically sets Mailbox Send Register bit 1 in the first mailbox to send the acknowledge back to the source core but, because Auto Link is also set, the mailbox automatically sets Mailbox Send Register bit 0 in the linked mailbox.

Interrupts and status Registers

When a core receives an IPCM interrupt, it determines which mailbox triggered it by reading the Masked Interrupt Status Register related to that interrupt line. Each Masked Interrupt Status Register contains up to 32 bits, each bit referring to a single mailbox.

If a core is using a mailbox in polled mode, it can use the Raw Interrupt Status Register to indicate which mailbox requires attention.

In Figure 2-4, each mailbox contains up to seven data registers to hold the message. Every mailbox instance with a single IPCM must have the same number of data registers.

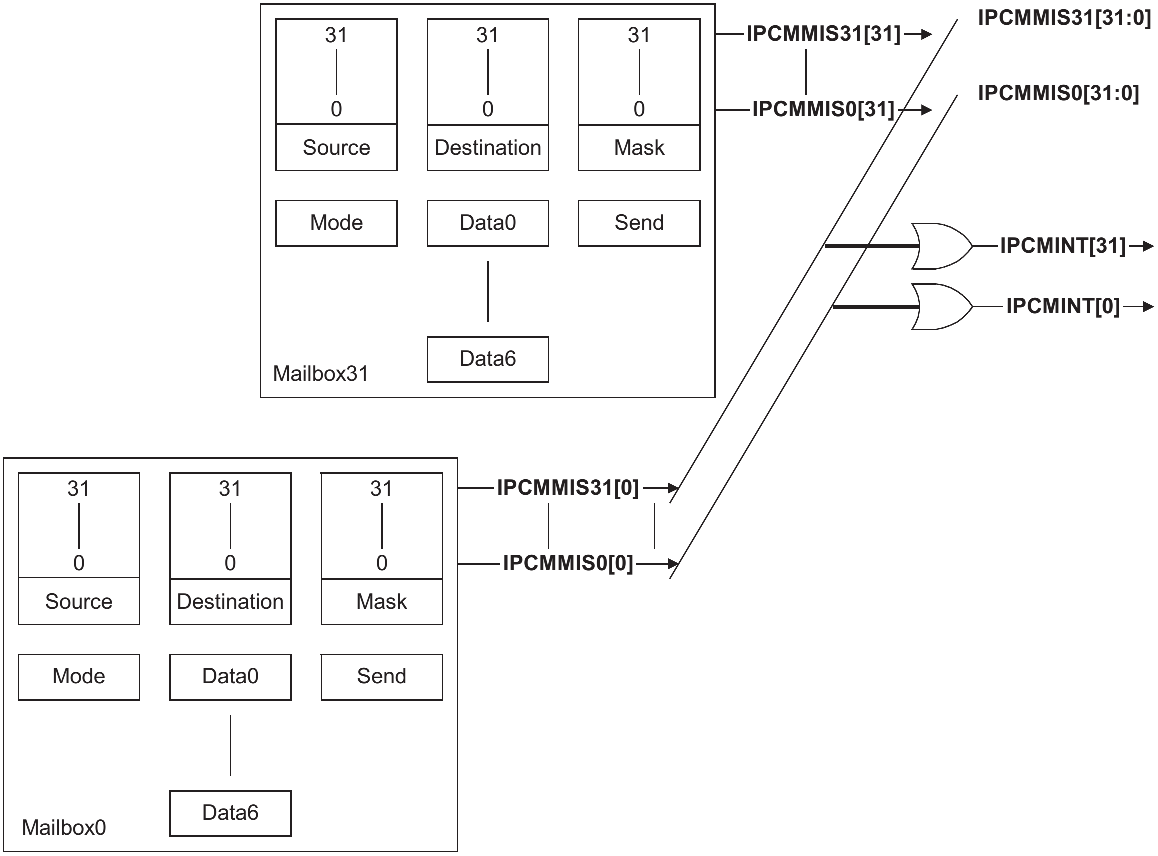


Figure 2-4 Mailbox interrupt mapping to IPCM interrupt outputs

Each mailbox can generate up to 32 interrupts, one for each Channel ID. The number of interrupts defines the number of bits in the Mailbox Source Register, Mailbox Destination Register, and Mailbox Mask Register. For example, in Figure 2-4, the IPCM has 32 interrupt outputs. Mailbox0 generates bit 0 of the **IPCMMIS0-31** buses, while Mailbox31 generates bit 31 of the **IPCMMIS0-31** buses.

Multiple mailboxes are grouped together as shown in Figure 2-4 on page 2-11 to form the 32-bit IPCM interrupt bus, **IPCMINT[31:0]**. All the interrupt bits from each mailbox relating to a single Channel ID are grouped together to form the masked interrupt status buses, **IPCMMIS0[31:0]** to **IPCMMIS31[31:0]**. The bits within these buses are then ORed together to form the IPCM interrupt bus, **IPCMINT[31:0]**.

Configuration Status Register

The three configurable parameters for the IPCM are:

- number of mailboxes, 1-32
- number of data registers per mailbox, 0-7
- number of interrupts, 1-32.

The configuration options that you choose define the read-only Configuration Status Register, enabling software to determine the IPCM configuration by reading this register. This enables a generic IPCM software driver to determine how to use each IPCM instance within a system.

To define the IPCM configuration, tie off the **MBOXNUM**, **INTNUM**, and **DATANUM** input pins as follows:

Number of mailboxes

Program the number of active mailboxes by tying off the **MBOXNUM** input bus (Table 2-2).

Table 2-2 Configuring number of mailboxes

Number of mailboxes	MBOXNUM
1	6'b000001
2	6'b000010
3	6'b000011
4	6'b000100
5	6'b000101
6	6'b000110
7	6'b000111
8	6'b001000
9	6'b001001

Table 2-2 Configuring number of mailboxes (continued)

Number of mailboxes	MBOXNUM
10	6'b001010
11	6'b001011
12	6'b001100
13	6'b001101
14	6'b001110
15	6'b001111
16	6'b010000
17	6'b010001
18	6'b010010
19	6'b010011
20	6'b010100
21	6'b010101
22	6'b010110
23	6'b010111
24	6'b011000
25	6'b011001
26	6'b011010
27	6'b011011
28	6'b011100
29	6'b011101
30	6'b011110
31	6'b011111
32	6'b100000

Note

Any setting of **MBOXNUM** other than the values shown in Table 2-2 on page 2-12 is unsupported.

Number of interrupts

Program the number of active interrupt outputs by tying off the **INTNUM** input bus (Table 2-3).

Table 2-3 Configuring number of interrupts

Number of mailboxes	INTNUM
1	6'b000001
2	6'b000010
3	6'b000011
4	6'b000100
5	6'b000101
6	6'b000110
7	6'b000111
8	6'b001000
9	6'b001001
10	6'b001010
11	6'b001011
12	6'b001100
13	6'b001101
14	6'b001110
15	6'b001111
16	6'b010000
17	6'b010001
18	6'b010010
19	6'b010011

Table 2-3 Configuring number of interrupts (continued)

Number of mailboxes	INTNUM
20	6'b010100
21	6'b010101
22	6'b010110
23	6'b010111
24	6'b011000
25	6'b011001
26	6'b011010
27	6'b011011
28	6'b011100
29	6'b011101
30	6'b011110
31	6'b011111
32	6'b100000

Note

Any setting of **INTNUM** other than the values shown in Table 2-3 on page 2-14 is unsupported.

Number of data registers in each mailbox

Program the number of active data registers in each mailbox by tying off the **DATANUM** input bus (Table 2-4).

Table 2-4 Configuring number of data registers

Number of mailboxes	DATANUM
0	3’b000
1	3'b001
2	3'b010
3	3'b011
4	3'b100
5	3'b101
6	3'b110
7	3'b111

——— **Note** ———

Any setting of **DATANUM** other than the values shown in Table 2-4 is unsupported.

Usage constraints

There are several valid use models for a mailbox and some constraints under which they can be used. Messages can be sent to:

Multiple cores If a message is sent to multiple cores, you must use the Auto Acknowledge feature and data must not be modified for the acknowledge. Destination cores must clear their interrupts by writing their Channel ID to the Destination Clear Register.

Single core If there is only a single destination core, the Auto Acknowledge mode is optional. If you disable the Auto Acknowledge mode, the acknowledge is optional, although an acknowledge normally happens, and the Mailbox Data Register can optionally be updated. When Auto Acknowledge is disabled, the destination core must clear its interrupt by clearing bit 0 of the Mailbox Send Register.

You can only use the Auto Link feature when there is an acknowledge. You can use the Auto Link feature with either:

Auto Acknowledge enabled

The mailbox automatically sets the acknowledge when the final destination core clears its interrupt.

Auto Acknowledge disabled

The destination core must send the acknowledge.

2.3 Examples of messaging

The following messaging examples are described in this section:

- *Messaging from Core0 to Core1*
- *Back-to-back messaging from Core0 to Core1* on page 2-20
- *Messaging from Core0 to Cores 1, 2, and 3 using Auto Acknowledge* on page 2-22
- *Auto Link messaging from Core0 to Core1 using Mailbox0 and Mailbox1* on page 2-24.

2.3.1 Messaging from Core0 to Core1

In this example system, there are two cores and four mailboxes. Core0 is the source core and Core1 is the destination core. Core0 uses Channel ID1 and Core1 uses Channel ID2. Core0 sends a message to Core1 using Mailbox0. This example assumes that the IPCM is not in integration test mode. Mailboxes 1-3 are inactive and Auto Acknowledge and Auto Link are disabled. Figure 2-5 shows the configuration.

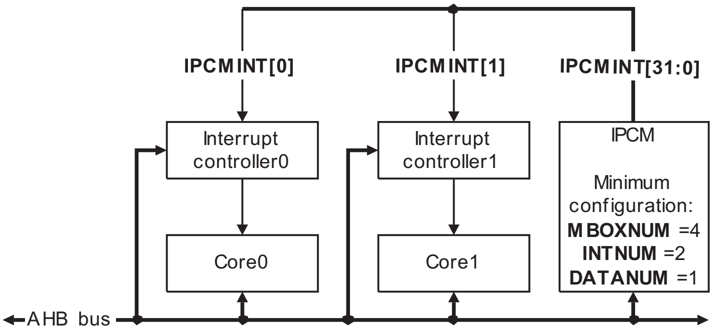


Figure 2-5 Configuration, messaging from Core0 to Core1

Figure 2-6 on page 2-19 shows the messaging sequence.

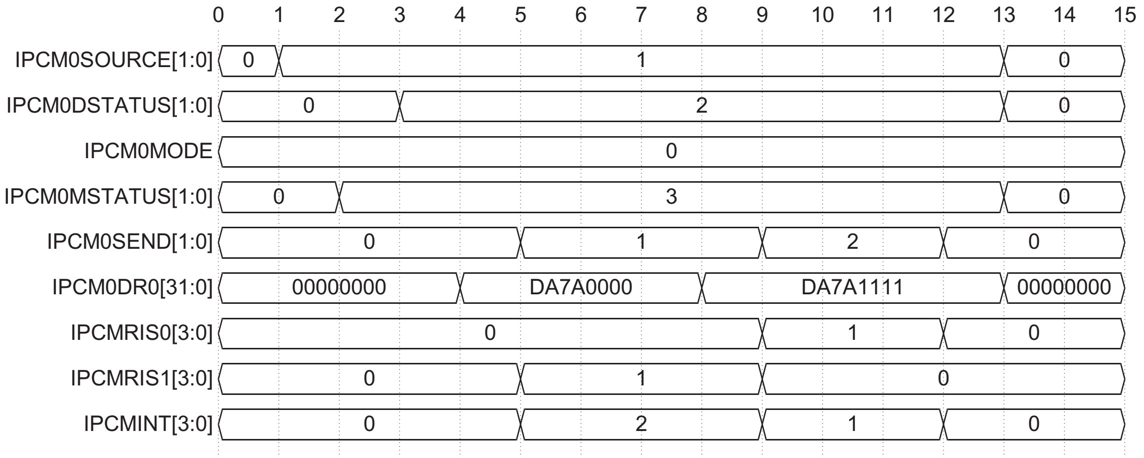


Figure 2-6 Messaging from Core0 to Core1

In this example, the following sequence occurs:

1. Core0 gains control of Mailbox 0 and identifies itself as the source core by setting bit 0 in the IPCM0SOURCE Register.
2. Core0 enables interrupts to Core0 and Core1 by setting bits 0 and 1 in the IPCM0MSTATUS Register.
3. Core0 defines the destination core by setting bit 1 in the IPCM0DSTATUS Register.
4. Core0 programs the data payload, `DA7A0000`.
5. Core0 sets Mailbox Send Register bit 0 to trigger the Mailbox0 interrupt to Core 1.
6. Core1 reads the IPCMRIS1 Register to determine which mailbox caused the interrupt. In this case, only Mailbox0 is indicated.
7. Core1 reads the data payload.
8. Core1 optionally updates the data payload with the Acknowledge data, `DA7A1111`.
9. Core1 clears bit 0 and sets bit 1 in the IPCM0SEND Register to clear its interrupt and provide the Manual Acknowledge interrupt back to Core0.
10. Core0 reads the IPCMRIS0 Register to determine which mailbox caused the interrupt. Again, only Mailbox0 is indicated.
11. Core0 reads the Acknowledge payload data.

12. Core0 clears bit 1 in the Mailbox Send Register to clear its interrupt.
13. Core0 releases ownership of the mailbox by clearing the IPCM0SOURCE Register, which in turn clears the IPCM0DSTATUS, IPCM0MSTATUS, and IPCM0DR0 Registers.

Note

Core0 can hold on to the mailbox to send another data message by not clearing the IPCM0SOURCE Register at step 13.

2.3.2 Back-to-back messaging from Core0 to Core1

In this example system, there are two cores and four mailboxes. Core0 is the source core and Core1 is the destination core. Core0 uses Channel ID1 and Core1 uses Channel ID2, as in *Back-to-back messaging from Core0 to Core1*. Core0 sends a message to Core1, obtains an acknowledge, and sends another message to Core1, which is also acknowledged. This example assumes that the IPCM is not in integration test mode. Mailboxes 1-3 are inactive and Auto Acknowledge and Auto Link are disabled. Figure 2-7 shows the configuration.

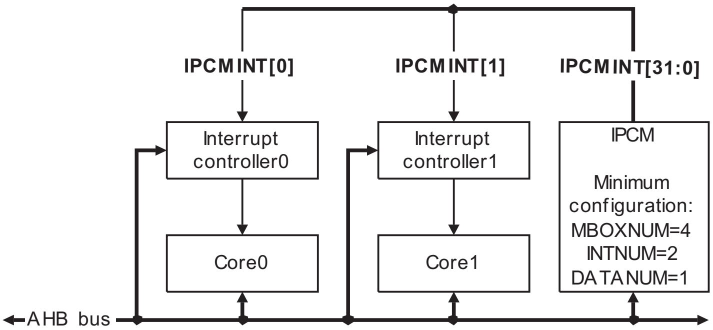


Figure 2-7 Configuration, back-to-back messaging from Core0 to Core1

Figure 2-8 on page 2-21 shows the messaging sequence.

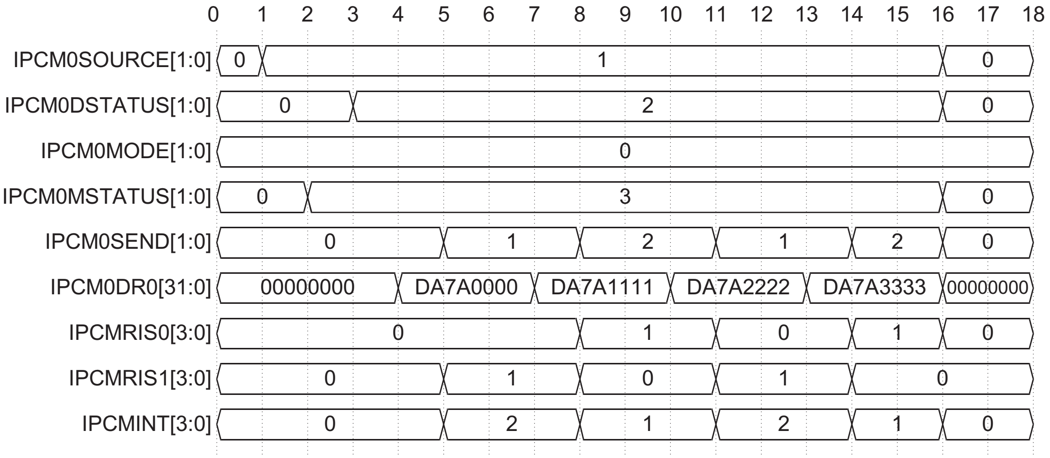


Figure 2-8 Back-to-back messaging from Core0 to Core1

In this example, the following sequence occurs:

1. Core0 gains control of Mailbox0 and identifies itself as the source core by setting bit 0 in the IPCM0SOURCE Register.
2. Core0 enables interrupts to Core0 and Core1 by setting bits 0 and 1 in the IPCM0MSTATUS Register.
3. Core0 defines the destination core by setting bit 1 in the IPCM0DSTATUS Register.
4. Core0 programs the data payload, `DA7A0000`.
5. Core0 sets bit 0 of the IPCM0SEND Register to send the interrupt to the destination core.
6. Core1 reads the IPCMRIS1 Register and reads the data payload.
7. Core1 optionally updates the data payload for the Acknowledge, `DA7A1111`.
8. Core1 clears bit 0 and sets bit 1 in the IPCM0SEND Register to provide the Manual Acknowledge back to Core0.
9. Core0 reads the IPCMRIS0 Register and reads the data payload.
10. Core0 programs the data payload for the next message, `DA7A2222`.
11. Core0 clears bit 1 and sets bit 0 of the IPCM0SEND Register to send the interrupt to the destination core.

12. Core1 reads the IPCMRIS1 Register and reads the data payload.
13. Core1 optionally updates the data payload for the Acknowledge, `DA7A3333`.
14. Core1 clears bit 0 and sets bit 1 in the IPCM0SEND Register to provide the Manual Acknowledge back to Core0.
15. Core0 reads the IPCMRIS0 Register and reads the data payload.
16. Core0 clears the interrupt and releases ownership of the mailbox by clearing the IPCM0SOURCE Register, which in turn clears the IPCM0DSTATUS, IPCM0MSTATUS, IPCM0SEND, and IPCM0DR0 Registers.

2.3.3 Messaging from Core0 to Cores 1, 2, and 3 using Auto Acknowledge

In this example system, there are four cores and four mailboxes:

- Core0 uses Channel ID1
- Core1 uses Channel ID2
- Core2 uses Channel ID4
- Core3 uses Channel ID8.

Core0 is the source core and sends a message to three destination cores, 1, 2, and 3. This example assumes that the IPCM is not in integration test mode. Mailboxes 1-3 are inactive and Auto Link is disabled. Figure 2-9 shows the configuration.

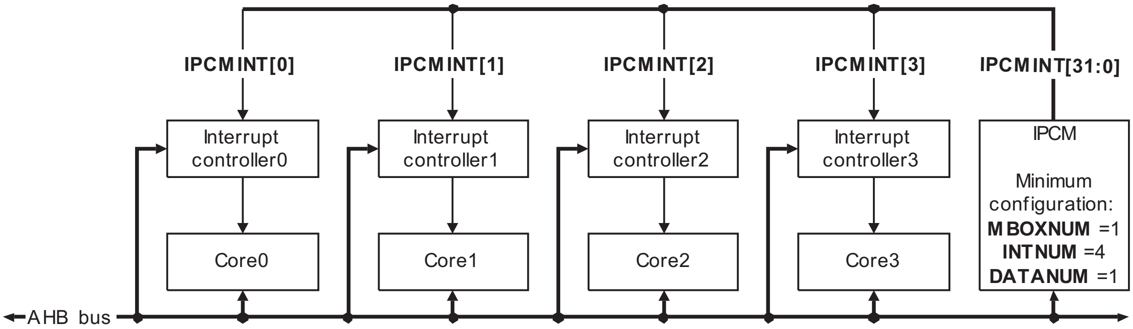


Figure 2-9 Configuration, messaging from Core0 to Cores 1, 2, and 3 using Auto Acknowledge

Figure 2-10 on page 2-23 shows the messaging sequence.

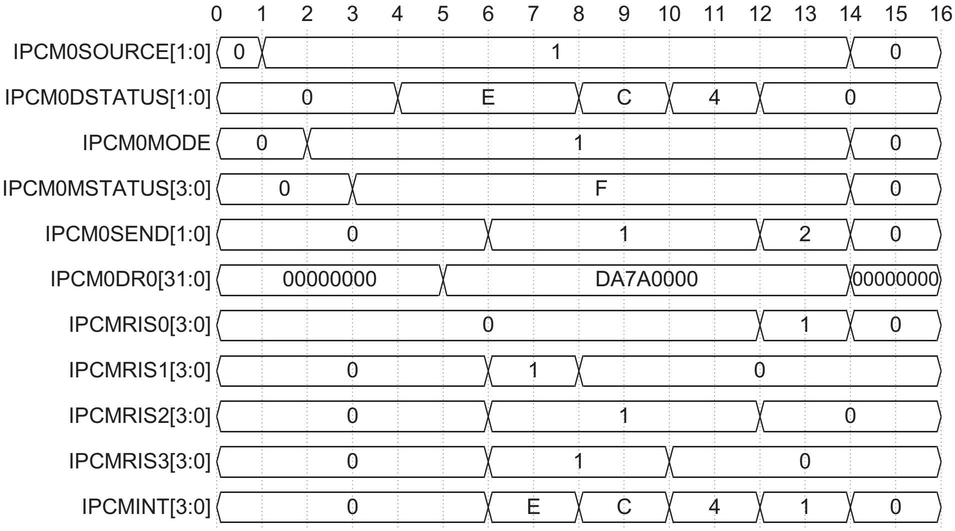


Figure 2-10 Messaging from Core0 to Cores 1, 2, and 3 using Auto Acknowledge

In this example, the following sequence occurs:

1. Core0 gains control of Mailbox0 and identifies itself as the source core by setting bit 0 in the IPCM0SOURCE Register.
2. Core0 sets Mailbox Mode Register bit 0 to put the mailbox into Auto Acknowledge mode.
3. Core0 enables interrupts to Core0, Core1, Core2, and Core3 by setting bits 0, 1, 2, and 3 in the IPCM0MSTATUS Register.
4. Core0 defines the destination cores by setting bits 1, 2, and 3 in the IPCM0DSTATUS Register.
5. Core0 programs the data payload, `DA7A0000`.
6. Core0 sets bit 0 of the IPCM0SEND Register to send the interrupts to the destination cores.
7. Core1 reads the IPCMRIS1 Register and reads the data payload.
8. Core1 clears bit 1 in the IPCM0DSTATUS Register.
9. Core3 reads the IPCMRIS3 Register and reads the data payload.
10. Core3 clears bit 3 in the IPCM0DSTATUS Register.

11. Core2 reads the IPCMRIS2 Register and reads the data payload.
12. Core2 clears bit 2 in the IPCM0DSTATUS Register. As the final Mailbox Destination Register bit is cleared, the mailbox automatically detects this, clears Mailbox Send Register bit 0 and sets Mailbox Send Register bit 1 to provide the Auto Acknowledge back to the source core, Core0. The data registers are not updated in Auto Acknowledge mode.
13. Core0 reads Status0 and reads the data payload.
14. Core0 clears the interrupt and releases ownership of the mailbox by clearing the IPCM0SOURCE register, which in turn clears the IPCM0SEND and IPCM0DR0 Registers.

Note

If Core0 has another message to send, it can maintain ownership of the mailbox by keeping the IPCM0SOURCE Register set, and updating the IPCM0DSTATUS, IPCM0MODE, IPCM0MSTATUS, and IPCM0DR0 Registers with the new message at step 14.

2.3.4 Auto Link messaging from Core0 to Core1 using Mailbox0 and Mailbox1

In this example system, there are two cores and four mailboxes. Core0 is the source core and Core1 is the destination core. Core0 uses Channel ID1 and Core1 uses Channel ID2. Core0 sets up Mailbox0 and Mailbox1 in Auto Link mode, and sends a message to Core1. Core1 responds to each interrupt separately and acknowledges both. Core0 only obtains an acknowledge interrupt when Core1 has finished with the final message. This example assumes that the IPCM has interrupts enabled and is not in integration test mode. Mailboxes 2-3 are inactive and Auto Acknowledge is disabled. Figure 2-11 on page 2-25 shows the configuration.

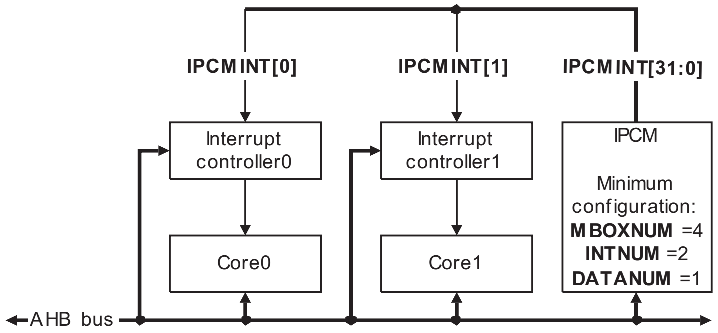


Figure 2-11 Configuration, Auto Link messaging from Core0 to Core1 using Mailbox0 and Mailbox1

Figure 2-12 shows the messaging sequence.

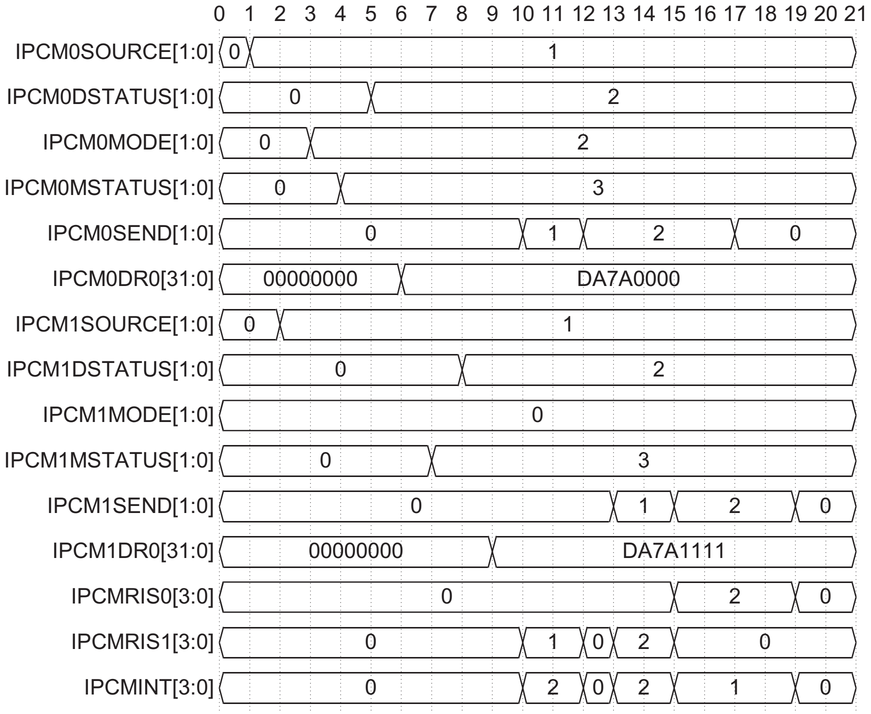


Figure 2-12 Auto Link messaging from Core0 to Core1 using Mailbox0 and Mailbox1

In this example, the following sequence occurs:

1. Core0 gains control of Mailbox0 and sets bit 0 in the IPCM0SOURCE Register.
2. Core0 gains control of Mailbox1 and sets bit 0 in the IPCM1SOURCE Register.
3. Core0 links Mailbox0 to Mailbox1 by setting bit 1 in the IPCM0MODE Register.
4. Core0 enables interrupts to Core0 and Core1 by setting bits 0 and 1 in the IPCM0MSTATUS Register.
5. Core0 defines the destination core of Mailbox0 by setting bit 1 in the IPCM0DSTATUS Register.
6. Core0 programs the data payload of Mailbox0 by setting the IPCM0DR0 Register to `DA7A0000`.
7. Core0 enables interrupts to Core0 and Core1 by setting bits 0 and 1 in the IPCM1MSTATUS Register.
8. Core0 defines the destination core of Mailbox1 by setting bit 1 in the IPCM1DSTATUS Register.
9. Core0 programs the data payload of Mailbox1 by setting Data1 to `DA7A1111`.
10. Core0 sets bit 1 in the IPCM0SEND Register to send the message in Mailbox0.
11. Core1 reads the IPCMRIS1 Register and reads the data payload in Mailbox0.
12. Core1 clears bit 0 and sets bit 1 in the IPCM0SEND Register to provide the Manual Acknowledge back to Core0.

    Note

    There is no acknowledge interrupt to Core0.

13. The message in Mailbox1 is automatically sent, triggered by bit 1 in the IPCM0SEND Register going HIGH and Auto Link mode being active.
14. Core1 reads the IPCMRIS1 Register and reads the data payload in Mailbox1.
15. Core1 clears bit 0 and sets bit 1 in the IPCM1SEND Register to provide the Manual Acknowledge back to Core0.

    Note

    This sends the acknowledge interrupt to Core0.

16. Core0 reads the IPCMRIS0 Register indicating that Mailbox1 has an acknowledge message. This indicates that the linked messages have all been sent. Core0 also reads the optional acknowledge data payload in Mailbox0.

17. Core0 clears bit 1 in the IPCM0SEND Register.

18. Core0 reads the optional acknowledge data payload in Core1.

19. Core0 clears bit 1 in the IPCM1SEND Register.

Chapter 3
Programmer's Model

This chapter describes the IPCM registers and gives details required when programming the device. It contains the following sections:

- *About the programmer's model* on page 3-2
- *Register summary* on page 3-6
- *Register descriptions* on page 3-12.

3.1 About the programmer's model

The following applies to the IPCM registers:

- The base address of the IPCM is not fixed and can be different for any particular system implementation. However, the offset of any particular register from the base address is fixed.
- Reserved or unused address locations must not be accessed because this can result in unpredictable behavior of the device.
- Reserved or unused bits of registers must be written as zero, and ignored on read unless otherwise stated in the relevant text.
- All register bits are reset to a 0 by a system or power-on reset unless otherwise stated in the relevant text.
- All registers support read and write accesses unless otherwise stated in the relevant text. A write updates the contents of a register and a read returns the contents of the register.

——— **Note** ———

Only Mailbox0 and the Interrupt Status registers for Interrupt0 are fully expanded for clarity. However, Mailboxes 1-31 at offsets `0x040-0x7FC`, and Interrupt 1-31 at offsets `0x808-0x8FC` also exist, depending on your configuration.

Because of the highly configurable nature of the IPCM, all the registers shown here might not be available in every configuration of the IPCM. Any writes to unavailable registers are ignored and any reads of unavailable registers return `0x00000000`.

MBOXNUM defines which Mailbox Registers are available. For example, when **MBOXNUM** is set to 1, only Mailbox0 Registers is available. When **MBOXNUM** is set to 32, all Mailbox Registers are available.

INTNUM defines the bit width of the IPCMxSOURCE, IPCMxDCLEAR, IPCMxDSET, IPCMxDSTATUS, IPCMxMCLEAR, IPCMxMSET, and IPCMxMSTATUS Registers. For example, when **INTNUM** is set to 1, the registers listed above are all only a single bit wide (bit0). Setting **INTNUM** to 32 sets the registers to 32 bits wide.

Secondly, **INTNUM** defines which IPCMRISx and IPCMMISx registers are available. For example, when **INTNUM** is set to 1, only the IPCMRIS0 and IPCMMIS0 Registers are available. When **INTNUM** is set to 32, all IPCMRIS0 to IPCMRIS31 Registers and IPCMMIS0 to IPCMMIS31 registers are available.

Finally, **INTNUM** also defines which interrupt outputs are active. Although **IPCMINT[31:0]** is always 32 bits wide, **INTNUM** defines which bits can be set. For example, when **INTNUM** is set to 1, only **IPCMINT[0]** can be set. When **INTNUM** is set to 32, all **IPCMINT[31:0]** bits can be set.

DATANUM defines which IPCMxDATAn registers are available, where x is defined by **MBOXNUM**. Setting **DATANUM** to 0 means there are no IPCMxDATAn Registers available. Setting **DATANUM** to 1 means the IPCMxDATA0 Registers are available. Setting **DATANUM** to 7, means all IPCMxDATA0 to IPCMDATA6 Registers are available.

Figure 3-1 on page 3-4 shows the IPCM register map.

Reserved for Interrupt6	0x830
Reserved for Interrupt5	0x828
Reserved for Interrupt4	0x820
Reserved for Interrupt3	0x818
Reserved for Interrupt2	0x810
Reserved for Interrupt1	0x808
Interrupt0	0x800
Reserved for Mailbox31	0x7C0
Reserved for Mailbox30	0x780
Reserved for Mailbox29	0x740
Reserved for Mailbox28	0x700
Reserved for Mailbox27	0x6C0
Reserved for Mailbox26	0x680
Reserved for Mailbox25	0x640
Reserved for Mailbox24	0x600
Reserved for Mailbox23	0x5C0
Reserved for Mailbox22	0x580
Reserved for Mailbox21	0x540
Reserved for Mailbox20	0x500
Reserved for Mailbox19	0x4C0
Reserved for Mailbox18	0x480
Reserved for Mailbox17	0x440
Reserved for Mailbox16	0x400
Reserved for Mailbox15	0x3C0
Reserved for Mailbox14	0x380
Reserved for Mailbox13	0x340
Reserved for Mailbox12	0x300
Reserved for Mailbox11	0x2C0
Reserved for Mailbox10	0x280
Reserved for Mailbox9	0x240
Reserved for Mailbox8	0x200
Reserved for Mailbox7	0x1C0
Reserved for Mailbox6	0x180
Reserved for Mailbox5	0x140
Reserved for Mailbox4	0x100
Reserved for Mailbox3	0x0C0
Reserved for Mailbox2	0x080
Reserved for Mailbox1	0x040
Mailbox0	0x000

PrimeCell Identification Register 3	0xFFC
PrimeCell Identification Register 2	0xFF8
PrimeCell Identification Register 1	0xFF4
PrimeCell Identification Register 0	0xFF0
Peripheral Identification Register 3	0xFEC
Peripheral Identification Register 2	0xFE8
Peripheral Identification Register 1	0xFE4
Peripheral Identification Register 0	0xFE0
Reserved	0xF08
Integration Test Output Register	0xF04
Integration Test Control Register	0xF00
Reserved	0x904
Configuration Status Register	0x900
Reserved for Interrupt31	0x8F8
Reserved for Interrupt30	0x8F0
Reserved for Interrupt29	0x8E8
Reserved for Interrupt28	0x8E0
Reserved for Interrupt27	0x8D8
Reserved for Interrupt26	0x8D0
Reserved for Interrupt25	0x8C8
Reserved for Interrupt24	0x8C0
Reserved for Interrupt23	0x8B8
Reserved for Interrupt22	0x8B0
Reserved for Interrupt21	0x8A8
Reserved for Interrupt20	0x8A0
Reserved for Interrupt19	0x898
Reserved for Interrupt18	0x890
Reserved for Interrupt17	0x888
Reserved for Interrupt16	0x880
Reserved for Interrupt15	0x878
Reserved for Interrupt14	0x870
Reserved for Interrupt13	0x868
Reserved for Interrupt12	0x860
Reserved for Interrupt11	0x858
Reserved for Interrupt10	0x850
Reserved for Interrupt9	0x848
Reserved for Interrupt8	0x840
Reserved for Interrupt7	0x838

Figure 3-1 IPCM register map

Figure 3-2 on page 3-5 shows the register map for Mailbox0.

Mailbox Data Register 6	0x03C
Mailbox Data Register 5	0x038
Mailbox Data Register 4	0x034
Mailbox Data Register 3	0x030
Mailbox Data Register 2	0x02C
Mailbox Data Register 1	0x028
Mailbox Data Register 0	0x024
Mailbox Send Registers	0x020
Mailbox Mask Status Registers	0x01C
Mailbox Mask Clear Registers	0x018
Mailbox Mask Set Registers	0x014
Mailbox Mode Registers	0x010
Mailbox Destination Status Registers	0x00C
Mailbox Destination Clear Registers	0x008
Mailbox Destination Set Registers	0x004
Mailbox Source Registers	0x000

Figure 3-2 Mailbox0 register map

Figure 3-3 shows the register map for each Interrupt0.

Raw Interrupt Status Registers	0x804
Masked Interrupt Status Registers	0x800

Figure 3-3 Interrupt0 register map

3.2 Register summary

All register addresses in the IPCM are fixed relative to the IPCM base address. Table 3-1 summarizes the IPCM registers.

Table 3-1 IPCM register summary

Name	Base offset	Type	Reset value	Description
IPCM0SOURCE	`0x000`	RW	`0x00000000`	See *Mailbox Source Registers* on page 3-12
IPCM0DSET	`0x004`	WO	-	See *Mailbox Destination Set Registers* on page 3-12
IPCM0DCLEAR	`0x008`	WO	-	See *Mailbox Destination Clear Registers* on page 3-13
IPCM0DSTATUS	`0x00C`	RO	`0x00000000`	See *Mailbox Destination Status Registers* on page 3-13
IPCM0MODE	`0x010`	RW	`0x0`	See *Mailbox Mode Registers* on page 3-13
IPCM0MSET	`0x014`	WO	-	See *Mailbox Mask Set Registers* on page 3-14
IPCM0MCLEAR	`0x018`	WO	-	See *Mailbox Mask Clear Registers* on page 3-15
IPCM0MSTATUS	`0x01C`	RO	`0x00000000`	See *Mailbox Mask Status Registers* on page 3-15
IPCM0SEND	`0x020`	RW	`0x0`	See *Mailbox Send Registers* on page 3-16
IPCM0DR0	`0x024`	RW	`0x00000000`	See *Mailbox Data Registers* on page 3-17
IPCM0DR1	`0x028`	RW	`0x00000000`	See *Mailbox Data Registers* on page 3-17
IPCM0DR2	`0x02C`	RW	`0x00000000`	See *Mailbox Data Registers* on page 3-17
IPCM0DR3	`0x030`	RW	`0x00000000`	See *Mailbox Data Registers* on page 3-17
IPCM0DR4	`0x034`	RW	`0x00000000`	See *Mailbox Data Registers* on page 3-17
IPCM0DR5	`0x038`	RW	`0x00000000`	See *Mailbox Data Registers* on page 3-17
IPCM0DR6	`0x03C`	RW	`0x00000000`	See *Mailbox Data Registers* on page 3-17
-	0x040-0x07C	-	-	Reserved for Mailbox1
-	0x080-0x0BC	-	-	Reserved for Mailbox2
-	0x0C0-0x0FC	-	-	Reserved for Mailbox3

Table 3-1 IPCM register summary (continued)

Name	Base offset	Type	Reset value	Description
-	0x100-0x13C	-	-	Reserved for Mailbox4
-	0x140-0x17C	-	-	Reserved for Mailbox5
-	0x180-0x1BC	-	-	Reserved for Mailbox6
-	0x1C0-0x1FC	-	-	Reserved for Mailbox7
-	0x200-0x23C	-	-	Reserved for Mailbox8
-	0x240-0x27C	-	-	Reserved for Mailbox9
-	0x280-0x2BC	-	-	Reserved for Mailbox10
-	0x2C0-0x2FC	-	-	Reserved for Mailbox11
-	0x300-0x33C	-	-	Reserved for Mailbox12
-	0x340-0x37C	-	-	Reserved for Mailbox13
-	0x380-0x3BC	-	-	Reserved for Mailbox14
-	0x3C0-0x3FC	-	-	Reserved for Mailbox15
-	0x400-0x43C	-	-	Reserved for Mailbox16
-	0x440-0x47C	-	-	Reserved for Mailbox17
-	0x480-0x4BC	-	-	Reserved for Mailbox18

Table 3-1 IPCM register summary (continued)

Name	Base offset	Type	Reset value	Description
-	0x4C0-0x4FC	-	-	Reserved for Mailbox19
-	0x500-0x53C	-	-	Reserved for Mailbox20
-	0x540-0x57C	-	-	Reserved for Mailbox21
-	0x580-0x5BC	-	-	Reserved for Mailbox22
-	0x5C0-0x5FC	-	-	Reserved for Mailbox23
-	0x600-0x63C	-	-	Reserved for Mailbox24
-	0x640-0x67C	-	-	Reserved for Mailbox25
-	0x680-0x6BC	-	-	Reserved for Mailbox26
-	0x6C0-0x6FC	-	-	Reserved for Mailbox27
-	0x700-0x73C	-	-	Reserved for Mailbox28
-	0x740-0x77C	-	-	Reserved for Mailbox29
-	0x780-0x7BC	-	-	Reserved for Mailbox30
-	0x7C0-0x7FC	-	-	Reserved for Mailbox31
IPCMMIS0	`0x800`	RO	`0x00000000`	See *Masked Interrupt Status Registers* on page 3-17
IPCMRIS0	`0x804`	RO	`0x00000000`	See *Raw Interrupt Status Registers* on page 3-18
-	0x808-0x80C	-	-	Reserved for Interrupt1

Table 3-1 IPCM register summary (continued)

Name	Base offset	Type	Reset value	Description
-	0x810-0x814	-	-	Reserved for Interrupt2
-	0x818-0x81C	-	-	Reserved for Interrupt3
-	0x820-0x824	-	-	Reserved for Interrupt4
-	0x828-0x82C	-	-	Reserved for Interrupt5
-	0x830-0x834	-	-	Reserved for Interrupt6
-	0x838-0x83C	-	-	Reserved for Interrupt7
-	0x840-0x844	-	-	Reserved for Interrupt8
-	0x848-0x84C	-	-	Reserved for Interrupt9
-	0x850-0x854	-	-	Reserved for Interrupt10
-	0x858-0x85C	-	-	Reserved for Interrupt11
-	0x860-0x864	-	-	Reserved for Interrupt12
-	0x868-0x86C	-	-	Reserved for Interrupt13
-	0x870-0x874	-	-	Reserved for Interrupt14
-	0x878-0x87C	-	-	Reserved for Interrupt15
-	0x880-0x884	-	-	Reserved for Interrupt16

Table 3-1 IPCM register summary (continued)

Name	Base offset	Type	Reset value	Description
-	0x888-0x88C	-	-	Reserved for Interrupt17
-	0x890-0x894	-	-	Reserved for Interrupt18
-	0x898-0x89C	-	-	Reserved for Interrupt19
-	0x8A0-0x8A4	-	-	Reserved for Interrupt20
-	0x8A8-0x8AC	-	-	Reserved for Interrupt21
-	0x8B0-0x8B4	-	-	Reserved for Interrupt22
-	0x8B8-0x8BC	-	-	Reserved for Interrupt23
-	0x8C0-0x8C4	-	-	Reserved for Interrupt24
-	0x8C8-0x8CC	-	-	Reserved for Interrupt25
-	0x8D0-0x8D4	-	-	Reserved for Interrupt26
-	0x8D8-0x8DC	-	-	Reserved for Interrupt27
-	0x8E0-0x8E4	-	-	Reserved for Interrupt28
-	0x8E8-0x8EC	-	-	Reserved for Interrupt29
-	0x8F0-0x8F4	-	-	Reserved for Interrupt30
-	0x8F8-0x8FC	-	-	Reserved for Interrupt31
IPCMCFGSTAT	0x900	RO	-	See *Configuration Status Register* on page 3-18

Table 3-1 IPCM register summary (continued)

Name	Base offset	Type	Reset value	Description
-	0x904-0xEFC	-	-	Reserved
IPCMTCR	`0xF00`	RW	`0x0`	See *Integration Test Control Register* on page 4-3
IPCMTOR	`0xF04`	RW	`0x00000000`	See *Integration Test Output Register* on page 4-3
-	0xF08-0xFDF	-	-	Reserved
IPCMPeriphID0	`0xFE0`	RO	`0x20`	See *Peripheral Identification Register 0* on page 3-20
IPCMPeriphID1	`0xFE4`	RO	`0x13`	See *Peripheral Identification Register 1* on page 3-21
IPCMPeriphID2	`0xFE8`	RO	`0x04`	See *Peripheral Identification Register 2* on page 3-21
IPCMPeriphID3	`0xFEC`	RO	`0x00`	See *Peripheral Identification Register 3* on page 3-21
IPCMPCellID0	`0xFF0`	RO	`0x0D`	See *PrimeCell Identification Register 0* on page 3-22
IPCMPCellID1	`0xFF4`	RO	`0xF0`	See *PrimeCell Identification Register 1* on page 3-23
IPCMPCellID2	`0xFF8`	RO	`0x05`	See *PrimeCell Identification Register 2* on page 3-23
IPCMPCellID3	`0xFFC`	RO	`0xB1`	See *PrimeCell Identification Register 3* on page 3-23

3.3 Register descriptions

This section describes the IPCM registers. Table 3-1 on page 3-6 provides cross references to individual registers.

Note

In the register names, x denotes a value of 0-31.

Mailbox Destination Registers and Mailbox Mask Registers have separate Set, Clear, and Status addresses.

3.3.1 Mailbox Source Registers

The read/write IPCMxSOURCE Registers define which core the message came from. The register is programmed with the Channel ID to identify which interrupt line to send the acknowledge interrupt through bit-wise encoding and can only be programmed to this value from `0x00000000`. When the register is programmed, it must be cleared to `0x00000000` before it can be reprogrammed.

Note

The software must ensure that IPCMxSOURCE is only programmed to a one-hot encoded value.

Table 3-2 lists the register bit assignments.

Table 3-2 IPCMxSOURCE Register bit assignments

Bit	Name	Function
[31:0]	Source	Set to define which core is the source and which interrupt line is asserted for the acknowledge interrupt

3.3.2 Mailbox Destination Set Registers

The write-only IPCMxDSET Registers set bits in the Mailbox Destination Registers. They can only be written to after the Mailbox Source Register is defined.

Table 3-3 lists the register bit assignments.

Table 3-3 IPCMxDSET Register bit assignments

Bit	Name	Function
[31:0]	Destination Set	Used to set bits in the IPCMxDSTATUS Registers

3.3.3 Mailbox Destination Clear Registers

The write-only IPCMxDCLEAR Registers clear bits in the Mailbox Destination Registers. They can only be written to after the Mailbox Source Register is defined.

Table 3-4 lists the register bit assignments.

Table 3-4 IPCMxDCLEAR Register bit assignments

Bit	Name	Function
[31:0]	Destination Clear	Used to clear bits in the Mailbox Destination Registers

3.3.4 Mailbox Destination Status Registers

The read-only IPCMxDSTATUS Registers contain the current status of the Mailbox Destination Registers.

When set, the Mailbox Destination Registers determine which cores to send the message to through bit-wise encoding using the Channel ID for each core. For cores that use multiple Channel IDs, only a single Channel ID is used per message.

The Mailbox Destination Registers are cleared in Auto Acknowledge Mode by destination cores to clear the mailbox interrupts to each core. When not in Auto Acknowledge mode, the Mailbox Destination Registers are only cleared by the source core when the mailbox is being reassigned. The Mailbox Destination Registers are cleared automatically by the mailbox regardless of which mode it is in when the Mailbox Source Register is cleared.

Table 3-5 lists the register bit assignments.

Table 3-5 IPCMxDSTATUS Register bit assignments

Bit	Name	Function
[31:0]	Destination Status	Gives the status of the Mailbox Destination Register. Defines which interrupt output to assert for the message.

3.3.5 Mailbox Mode Registers

The read/write IPCMxMODE Registers define how the mailbox is used. The registers can only be written to when the mailbox is assigned, indicated by a bit in the Mailbox Source Register being set.

The Auto Acknowledge bit provides an acknowledge interrupt back to the source core when the Mailbox Destination Register has been cleared. The Auto Acknowledge indicates when all cores that have received a message have cleared their interrupts. The Auto Acknowledge bit must always be set when there is more than one destination core.

The Auto Link bit links adjacent mailboxes together to enable multiple messages to be sent sequentially by the source core without the requirement for the source core to be interrupted between messages. Instead of an acknowledge interrupt being sent back to the source core, which can be done manually by a single destination core or automatically using Auto Acknowledge, the linked mailbox message is sent. The order of linking is fixed. Mailbox 0 links to Mailbox 1, which can link to Mailbox 2, up to Mailbox 31.

The IPCMxMODE Registers are cleared when the Mailbox Source Register is cleared.

Figure 3-4 shows the register bit assignments.

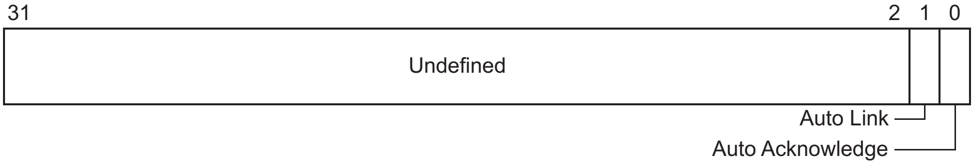


Figure 3-4 IPCMxMODE Register bit assignments

Table 3-6 lists the register bit assignments.

Table 3-6 IPCMxMODE Register bit assignments

Bit	Name	Function
[31:2]	-	Read undefined. Write as zero.
[1]	Auto Link	Set to enable Auto Link.
[0]	Auto Acknowledge	Set to enable Auto Acknowledge.

3.3.6 Mailbox Mask Set Registers

The write-only IPCMxMSET Registers set bits in the Mailbox Mask Registers. They can only be written to after the Mailbox Source Register is defined.

Table 3-7 lists the register bit assignments.

Table 3-7 IPCMxMSET Register bit assignments

Bit	Name	Function
[31:0]	Mask Set	Used to set bits in the Mailbox Mask Register

3.3.7 Mailbox Mask Clear Registers

The write-only IPCMxMCLEAR Registers clear bits in the Mailbox Mask Registers. They can only be written to after the Mailbox Source Register is defined.

Table 3-8 lists the register bit assignments.

Table 3-8 IPCMxMCLEAR Register bit assignments

Bit	Name	Function
[31:0]	Mask Clear	Used to clear bits in the Mailbox Mask Register

3.3.8 Mailbox Mask Status Registers

The read-only IPCMxMSTATUS Registers contain the current status of the Mailbox Mask Registers. Each core is assigned its own bit.

When set, the Mailbox Mask Registers enable the interrupts to each core through bit-wise encoding for each of the Channel IDs. These bits reset to 0, disabling the interrupts.

When cleared, the Mailbox Mask Registers disable the interrupts, enabling the cores to use polling rather than interrupts for messaging.

The Mailbox Mask Registers are all cleared when the Mailbox Source Register is cleared.

Table 3-9 lists the register bit assignments.

Table 3-9 IPCMxMSTATUS Register bit assignments

Bit	Name	Function
[31:0]	Mask Status	Gives the status of the Mailbox Mask Registers. For each bit position: 1 = Mailbox interrupt enabled 0 = Mailbox interrupt disabled, polling used instead.

3.3.9 Mailbox Send Registers

The read/write IPCMxSEND Registers send the message to either the source or destination cores.

The Mailbox Send Register bits can only be written to after the Mailbox Source Register is defined:

- setting bit 0 generates an interrupt to the destination core(s)
- setting bit 1 generates an interrupt to the source core.

Note

Setting both bits 0 and 1 is not valid and can give unpredictable results. Clearing any send bit clears the interrupt generated by that mailbox.

In Auto Acknowledge mode, when the Mailbox Destination Status Register changes from being non-zero to zero and the Mailbox Send Register currently contains 01, the mailbox automatically changes the register to 10, triggering the Auto Acknowledge interrupt back to the source core.

The Mailbox Send Registers are cleared when the Mailbox Source Register is cleared.

Figure 3-5 shows the register bit assignments.

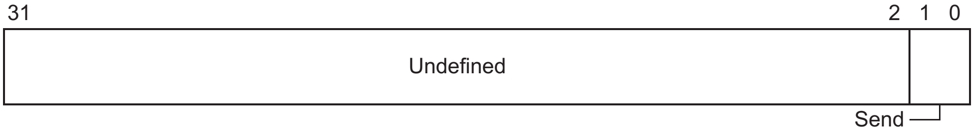


Figure 3-5 IPCMxSEND Register bit assignments

Table 3-10 lists the register bit assignments.

Table 3-10 IPCMxSEND Register bit assignments

Bit	Name	Function
[31:2]	-	Read undefined. Write as zero.
[1:0]	Send	Send message: 00 = inactive 01 = send message to destination core(s) 10 = send message to source core 11 = invalid, unpredictable behavior

3.3.10 Mailbox Data Registers

The read/write IPCMxDR0-6 Registers hold the message. The Mailbox Data Registers can only be written to after the Mailbox Source Register is defined and are cleared when the Mailbox Source Register is cleared.

Table 3-11 lists the register bit assignments.

Table 3-11 IPCMxDR0-6 Register bit assignments

Bit	Name	Function
[31:0]	Data	Message data

3.3.11 Masked Interrupt Status Registers

The read-only IPCMMISx Registers contain the current mailbox status for every interrupt identified by the address encoding. This enables each core to read a single register to determine which mailbox caused the interrupt. For example, if Core0 is mapped to Channel ID0, it reads IPCMMIS0 to determine which mailboxes require attention.

Figure 3-6 on page 3-18 shows how Mailbox0 status is presented to Core0 through the use of two status registers, IPCMMIS0 and IPCMRIS0.

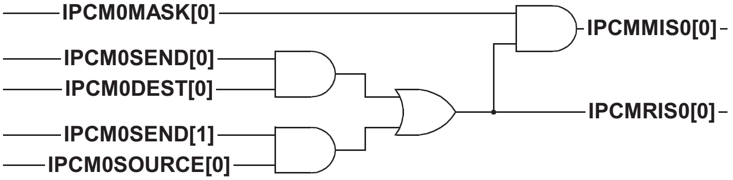


Figure 3-6 Mailbox status

The Masked Interrupt Status Registers identify which mailbox triggered the interrupt. This value is the logical AND of the raw interrupt status with the Mailbox Mask Status Registers. All Masked Interrupt Status Register outputs are ORed together to form the **IPCMINT[31:0]** interrupt output bus.

Table 3-12 lists the register bit assignments.

Table 3-12 IPCMMISx Register bit assignments

Bit	Name	Function
[31:0]	MaskIntStat	Masked interrupt status

3.3.12 Raw Interrupt Status Registers

The read-only IPCMRISx Registers indicate the unmasked interrupt status of each mailbox for each core.

Table 3-13 lists the register bit assignments.

Table 3-13 IPCMRISx Register bit assignments

Bit	Name	Function
[31:0]	RawIntStat	Raw interrupt status

3.3.13 Configuration Status Register

The read-only IPCMCFGSTAT Register indicates the hardware configuration options chosen for implementation of the IPCM.

Figure 3-7 on page 3-19 shows the register bit assignments.

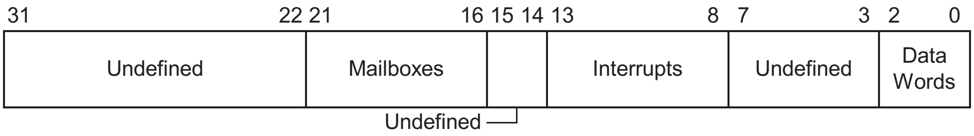


Figure 3-7 IPCMCFGSTAT Register bit assignments

Table 3-14 lists the register bit assignments.

Table 3-14 IPCMCFGSTAT Register bit assignments

Bit	Name	Function
[31:22]	-	Read undefined
[21:16]	Mailboxes	Returns the value of the **MBOXNUM** input pins
[15:14]	-	Read undefined
[13:8]	Interrupts	Returns the value of the **INTNUM** input pins
[7:3]	-	Read undefined
[2:0]	Data Words	Returns the value of the **DATANUM** input pins

3.3.14 Peripheral Identification Registers

The IPCMPeriphID0-3 Registers are four 8-bit registers, that span address locations `0xFE0-0xFEC`. You can conceptually treat the registers as a single 32-bit register. The read-only registers provide the following options of the peripheral:

PartNumber[11:0] This is used to identify the peripheral. The product code `0x320` is used for the IPCM.

DesignerID[19:12] This is the identification of the designer. ARM Limited is `0x41` (ASCII A).

Revision[23:20] This is the revision number of the peripheral. The revision number starts from 0 and is revision dependent.

Configuration[31:24]

This is the configuration option of the peripheral. The configuration value is 0.

Figure 3-8 on page 3-20 shows the register bit assignments.

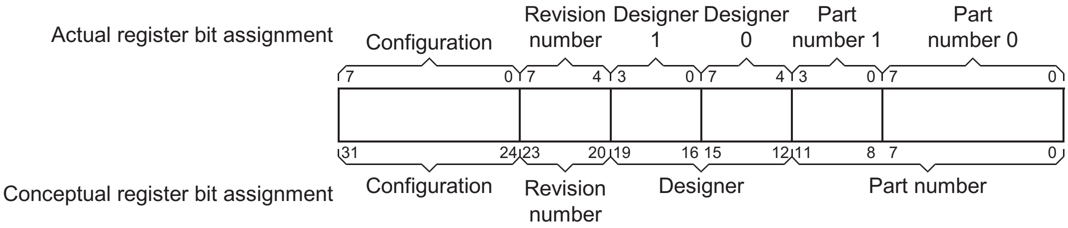


Figure 3-8 Peripheral Identification Register bit assignments

Note

When you design a system memory map then you must remember that the register has a 4KB-memory footprint. All memory accesses to the peripheral identification registers must be 32-bit, using the `LDR` and `STR` instructions.

The Peripheral Identification Registers are described in the following subsections:

- *Peripheral Identification Register 0*
- *Peripheral Identification Register 1* on page 3-21
- *Peripheral Identification Register 2* on page 3-21
- *Peripheral Identification Register 3* on page 3-21.

Peripheral Identification Register 0

The hard-coded IPCMPeriphID0 Register defines the reset value. Table 3-15 lists the bit assignments for the IPCMPeriphID0 Register.

Table 3-15 IPCMPeriphID0 Register bit assignments

Bits	Name	Description
[31:8]	-	Read undefined
[7:0]	PartNumber0	These bits read back as `0x20`

Peripheral Identification Register 1

The hard-coded IPCMPeriphID1 Register defines the reset value. Table 3-16 lists the bit assignments for the IPCMPeriphID1 Register.

Table 3-16 IPCMPeriphID1 Register bit assignments

Bits	Name	Description
[31:8]	-	Read undefined
[7:4]	Designer0	These bits read back as `0x1`
[3:0]	PartNumber1	These bits read back as `0x3`

Peripheral Identification Register 2

The hard-coded IPCMPeriphID2 Register defines the reset value. Table 3-17 lists the bit assignments for the IPCMPeriphID2 Register.

Table 3-17 IPCMPeriphID2 Register bit assignments

Bits	Name	Description
[31:8]	-	Read undefined
[7:4]	Revision	These bits read back as `0x0`
[3:0]	Designer1	These bits read back as `0x4`

Peripheral Identification Register 3

The hard-coded IPCMDPeriphID3 Register defines the reset value. Table 3-18 lists the bit assignments for the IPCMPeriphID3 Register.

Table 3-18 IPCMPeriphID3 Register bit assignments

Bits	Name	Description
[31:8]	-	Read undefined
[7:0]	Configuration	These bits read back as `0x00`

3.3.15 PrimeCell Identification Registers

The IPCMPCellID0-3 Registers are four 8-bit registers, that span address locations `0xFF0-0xFFC`. You can conceptually treat the registers as a single 32-bit register. The register is used as a standard cross-peripheral identification system.

Figure 3-9 shows the register bit assignments.

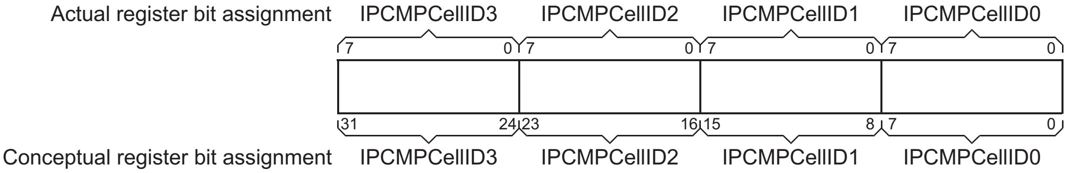


Figure 3-9 PrimeCell Identification Register bit assignments

The four PrimeCell Identification Registers are described in the following subsections:

- *PrimeCell Identification Register 0*
- *PrimeCell Identification Register 1* on page 3-23
- *PrimeCell Identification Register 2* on page 3-23
- *PrimeCell Identification Register 3* on page 3-23.

PrimeCell Identification Register 0

The hard-coded IPCMPCellID0 Register defines the reset value. Table 3-19 lists the bit assignments for the IPCMPCellID0 Register.

Table 3-19 IPCMPCellID0 Register bit assignments

Bits	Name	Description
[31:8]	-	Read undefined
[7:0]	IPCMPCellID0	These bits read back as `0x0D`

PrimeCell Identification Register 1

The hard-coded IPCMPCellID1 Register defines the reset value. Table 3-20 lists the bit assignments for the IPCMPCellID1 Register.

Table 3-20 IPCMPCellID1 Register bit assignments

Bits	Name	Description
[31:8]	-	Read undefined
[7:0]	IPCMPCellID1	These bits read back as `0xF0`

PrimeCell Identification Register 2

The hard-coded IPCMPCellID2 Register defines the reset value. Table 3-21 lists the bit assignments for the IPCMPCellID2 Register.

Table 3-21 IPCMPCellID2 Register bit assignments

Bits	Name	Description
[31:8]	-	Read undefined
[7:0]	IPCMPCellID2	These bits read back as `0x05`

PrimeCell Identification Register 3

The hard-coded IPCMPCellID3 Register defines the reset value. Table 3-22 lists the bit assignments for the IPCMPCellID3 Register.

Table 3-22 IPCMPCellID3 Register bit assignments

Bits	Name	Description
[31:8]	-	Read undefined
[7:0]	IPCMPCellID3	These bits read back as `0xB1`

Chapter 4
Programmer's Model for Test

This chapter describes the additional logic for functional verification and production testing. It contains the following sections:

4.1 Scan testing

The IPCM enables:

- the automatic insertion of scan test cells
- the use of *Automatic Test Pattern Generation* (ATPG).

This is the recommended method of manufacturing test.

During scan testing, ensure that the **SCANENABLE** signal is driven HIGH. For normal use ensure that **SCANENABLE** is driven LOW.

4.2 Test registers

The input configuration pin **INTNUM** defines the bit width of the IPCMTOR register. For example, when **INTNUM** is set to 1, IPCMTOR is only a single bit wide (bit 0). Setting **INTNUM** to 32 sets IPCMTOR to 32 bits wide.

The IPCM test registers are memory-mapped as shown in *IPCM register map* on page 3-4 and Table 3-1 on page 3-6. The address offset is from the base address.

4.2.1 Integration Test Control Register

The read/write IPCMTCR Register controls the IPCM integration test mode. When ITEN=1, the IPCM is placed in integration test mode. Figure 4-1 shows and Table 4-1 lists the register bit assignments.

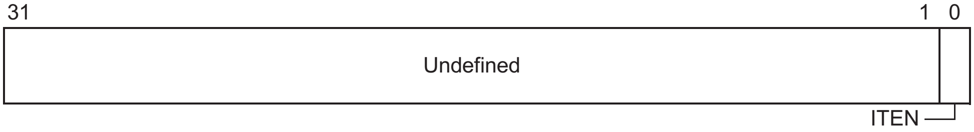


Figure 4-1 IPCMTCR Register bit assignments

Table 4-1 IPCMTCR Register bit assignments

Bit	Name	Function
[31:1]	-	Read undefined. Write as zero.
[0]	ITEN	Integration test enable: 0 = integration test mode disabled 1 = integration test mode enabled.

4.2.2 Integration Test Output Register

The read/write IPCMTOR Register enables the output port signals of the IPCM to be driven directly rather than from their normal internal logic source when in integration test mode, that is, when ITEN=1 in the IPCMTCR Register. Table 4-2 lists the register bit assignments.

Table 4-2 IPCMTOR Register bit assignments

Bit	Name	Function
[31:0]	IntTest	**IPCMINT[31:0]** output

Appendix A
Signal Descriptions

This appendix describes the signals that interface with the IPCM. It contains the following sections:

- *AMBA AHB signals* on page A-2
- *Non-AMBA signals* on page A-3.

A.1 AMBA AHB signals

Table A-1 lists the AMBA AHB common signals.

Table A-1 AMBA AHB common signals

Name	Type	Source/destination	Description
HCLK	Input	Clock controller	Clock input for all IPCM flops
HRESETn	Input	Reset controller	AHB bus reset, active LOW

Table A-2 lists the AMBA AHB slave signals.

Table A-2 AMBA AHB slave signals

Name	Type	Source/destination	Description
HADDR[11:2]	Input	Send or receive core AHB	System address bus
HRDATA[31:0]	Output	Send or receive core AHB	Read data bus
HREADY	Input	Send or receive core AHB	Transfer completed input. When HIGH, this signal indicates that a transfer has finished on the bus.
HREADYOUT	Output	Send or receive core AHB	Transfer done output. When HIGH, this signal indicates that a transfer has finished on the bus. This signal can be driven LOW to extend a transfer. The IPCM is always zero wait state, therefore this signal is always driven HIGH.
HRESP[1:0]	Output	Send or receive core AHB	The transfer response provides additional information on the status of a transfer. The IPCM always provides an OKAY response.
HSEL	Input	Send or receive core AHB	Slave select signal for IPCM control and status registers
HSIZE[2:0]	Input	Send or receive core AHB	Transfer size signal. This signal indicates the size of the current transfer, which can be byte (8-bit), halfword (16-bit), or word (32-bit). The IPCM only supports 32-bit transfers.
HTRANS	Input	Send or receive core AHB	Indicates the type of the current transfer, which can be NONSEQUENTIAL, SEQUENTIAL, IDLE, or BUSY. The IPCM only uses **HTRANS[1]**.
HWDATA[31:0]	Input	Send or receive core AHB	Write data bus
HWRITE	Input	Send or receive core AHB	Transfer direction signal. When HIGH, this signal indicates a write and, when LOW, a read

A.2 Non-AMBA signals

Table A-4 lists the IPCM configuration signals.

Table A-3 IPCM configuration signals

Name	Type	Source/destination	Description
DATANUM[2:0]	Input	Tied off	Number of data registers in each mailbox
INTNUM[5:0]	Input	Tied off	Number of interrupts
MBOXNUM[5:0]	Input	Tied off	Number of mailboxes

Table A-4 shows the IPCM interrupt signal.

Table A-4 IPCM interrupt signals

Name	Type	Source/destination	Description
IPCMINT[31:0]	Output	Vectored interrupt controller	IPCM interrupt, active HIGH

Note

The configuration of the IPCM is defined by tieing off the **MBOXNUM**, **INTNUM**, and **DATANUM** input pins, as described in *Configuration Status Register* on page 2-12.

Table A-5 lists the scan test signals.

Table A-5 Scan test signals

Name	Type	Source/destination	Description
SCANENABLE	Input	Scan controller	Scan enable
SCANINHCLK	Input	Scan controller	Scan data input for **HCLK** domain
SCANOUTHCLK	Output	Scan controller	Scan data output for **HCLK** domain

Glossary

This glossary describes some of the terms used in ARM manuals. Where terms can have several meanings, the meaning presented here is intended.

Advanced eXtensible Interface (AXI)

This is a bus protocol that supports separate address/control and data phases, unaligned data transfers using byte strobes, burst-based transactions with only start address issued, separate read and write data channels to enable low-cost DMA, ability to issue multiple outstanding addresses, out-of-order transaction completion, and easy addition of register stages to provide timing closure.The AXI protocol also includes optional extensions to cover signaling for low-power operation.

AXI is targeted at high performance, high clock frequency system designs and includes a number of features that make it very suitable for high speed sub-micron interconnect.

Advanced High-performance Bus (AHB)

The AMBA Advanced High-performance Bus system connects embedded processors such as an ARM core to high-performance peripherals, DMA controllers, on-chip memory, and interfaces. It is a high-speed, high-bandwidth bus that supports multi-master bus management to maximize system performance.

See also Advanced Microcontroller Bus Architecture.

Advanced Microcontroller Bus Architecture (AMBA)

AMBA is the ARM open standard for multi-master on-chip buses, capable of running with multiple masters and slaves. It is an on-chip bus specification that details a strategy for the interconnection and management of functional blocks that make up a System-on-Chip (SoC). It aids in the development of embedded processors with one or more CPUs or signal processors and multiple peripherals. AMBA complements a reusable design methodology by defining a common backbone for SoC modules. AHB conforms to this standard.

Advanced Peripheral Bus (APB)

The AMBA Advanced Peripheral Bus is a simpler bus protocol than AHB. It is designed for use with ancillary or general-purpose peripherals such as timers, interrupt controllers, UARTs, and I/O ports. Connection to the main system bus is through a system-to-peripheral bus bridge that helps to reduce system power consumption.

See also Advanced High-performance Bus.

AHB *See* Advanced High-performance Bus.

AMBA *See* Advanced Microcontroller Bus Architecture.

APB *See* Advanced Peripheral Bus.

ATPG *See* Automatic Test Pattern Generation.

Automatic Test Pattern Generation (ATPG)

The process of automatically generating manufacturing test vectors for an ASIC design, using a specialized software tool.

AXI *See* Advanced eXtensible Interface.

Byte An 8-bit data item.

Core A core is that part of a processor that contains the ALU, the datapath, the general-purpose registers, the Program Counter, and the instruction decode and control circuitry.

DNM *See* Do Not Modify.

Do Not Modify (DNM)

In Do Not Modify fields, the value must not be altered by software. DNM fields read as Unpredictable values, and must only be written with the same value read from the same field on the same processor.

Halfword A 16-bit data item.

Processor A processor is the circuitry in a computer system required to process data using the computer instructions. It is an abbreviation of microprocessor. A clock source, power supplies, and main memory are also required to create a minimum complete working computer system.

Reserved A field in a control register or instruction format is reserved if the field is to be defined by the implementation, or produces Unpredictable results if the contents of the field are not zero. These fields are reserved for use in future extensions of the architecture or are implementation-specific. All reserved bits not used by the implementation must be written as 0 and read as 0.

Unpredictable For reads, the data returned when reading from this location is unpredictable. It can have any value. For writes, writing to this location causes unpredictable behavior, or an unpredictable change in device configuration. Unpredictable instructions must not halt or hang the processor, or any part of the system.

Word A 32-bit data item.

Index