



Split Transaction Feature for the Intel[®] PXA27x Processor Family

Application Note

October 2004



INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The Intel® PXA27x processor may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

MPEG is an international standard for video compression/decompression promoted by ISO. Implementations of MPEG CODECs, or MPEG enabled platforms may require licenses from various entities, including Intel Corporation.

This document and the software described in it are furnished under license and may only be used or copied in accordance with the terms of the license. The information in this document is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Intel Corporation. Intel Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this document or any software that may be provided in association with this document. Except as permitted by such license, no part of this document may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without the express written consent of Intel Corporation.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature may be obtained by calling 1-800-548-4725 or by visiting Intel's website at <http://www.intel.com>.

AlertVIEW, AnyPoint, AppChoice, BoardWatch, BunnyPeople, CablePort, Celeron, Chips, CT Connect, CT Media, Dialogic, DM3, EtherExpress, ETOX, FlashFile, i386, i486, i960, iCOMP, InstantIP, Intel, Intel logo, Intel386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2, Intel Create & Share, Intel GigaBlade, Intel InBusiness, Intel Inside, Intel Inside logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel Play, Intel Play logo, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel TeamStation, Intel Xeon, Intel XScale, IPLink, Itanium, LANDesk, LanRover, MCS, MMX, MMX logo, Optimizer logo, OverDrive, Paragon, PC Dads, PC Parents, PDCharm, Pentium, Pentium II Xeon, Pentium III Xeon, Performance at Your Command, RemoteExpress, Shiva, SmartDie, Solutions960, Sound Mark, StorageExpress, The Computer Inside., The Journey Inside, TokenExpress, Trillium, VoiceBrick, Vtune, and Xircor are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © Intel Corporation, 2004. All Rights Reserved.

1.0 Introduction

The Intel® PXA27x Processor Family (PXA27x processor) contains a device called a direct-memory access (DMA) controller and bridge unit. Most developers understand the DMA portion of the unit but rarely think about the bridge portion of the unit. The DMA controller can be set up to move blocks of data to and from peripherals without intervention from the core. The bridge performs read/writes to and from the peripheral units that the core initiated, but the core can be released to continue processing the subsequent tasks while the read/write is occurring across the internal buses.

Developers using the Intel® PXA27x processor often consider the internal DMA controller only as a controller to initiate direct-memory access between peripherals and are not aware that the DMA controller can serve as a bridge for read and writes issued from the Intel XScale® core.

Some developers may think that once a write is issued to a peripheral they can issue another read or write command to another peripheral, but they are not aware of the potential for problems because of the split-transaction feature.

Other developers may not be aware of the difference between a system-bus peripheral and peripheral-bus peripheral.

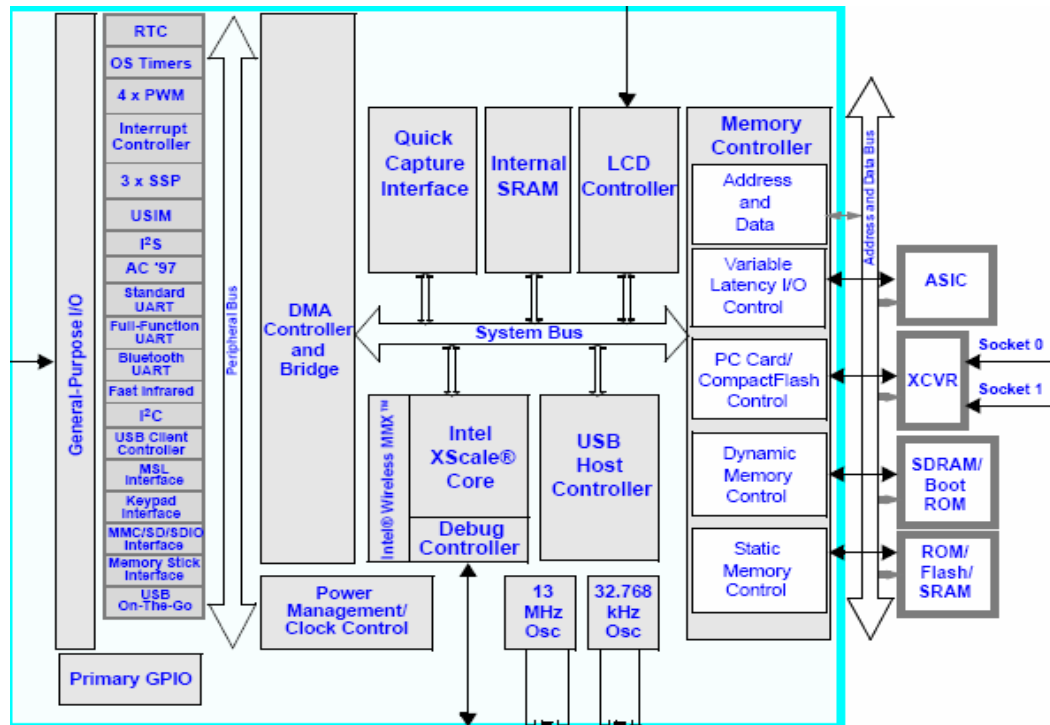
This application note addresses all of these topics so that developers using the PXA27x processor are fully aware of the DMA bridge feature and the split-transaction feature, and understand how to ensure safe split transactions and how to explore the maximum bus utilization.

2.0 Description

2.1 Bulverde Architecture

Figure 1 is a block diagram for a typical system using an Intel® PXA27x processor.

Figure 1. Intel® PXA27x Processor Block Diagram for a Typical System



The PX27x system bus allows access to the following clients:

- Intel XScale® core
- USB host controller
- LCD controller
- Internal SRAM
- Camera interface
- DMA controller and bridge
- Memory controller

The system bus can be programmed to operate at a maximum frequency of 208 MHz to achieve the processor's highest level of performance. These system-bus clients are all high-speed devices that require fast access time. Rather than using a three-state approach, the bus is multiplexed and clients can request the bus without any limitations.

More peripherals are accessible through the peripheral bus, which itself is accessed through the DMA bridge. Regardless of the system-bus speed, the peripheral bus always runs at 26 MHz to accommodate low-speed peripherals.

The interrupt controller is accessible through the low-speed peripheral bus. Because interrupt latency is critical for system performance, there is another interface exposed allowing the Intel XScale® core to access interrupt controller via Coprocessor 6 quickly. With the exception of the ICCR and the IPR, all interrupt controller registers are accessible through both the peripheral bus and Coprocessor 6 interfaces.

2.2 What is a Split Transaction?

Processor accesses to the peripheral bus are by default *split* or *posted* operations (refer to Section 5.5.11, “DPCSR[BrgSplit]” in the *Intel® PXA27x Processor Family Developers Manual*). In other words, a read operation requires two system-bus transactions: (1) an initial transaction to send the request, and (2) a separate data transfer to return the read data. During the gap between these two transactions, the system bus is relinquished for use by other devices. However, the core is stalled while waiting for the read transaction to complete; therefore, the core can not inadvertently process something that is out of order.

For a write operation, the system-bus transaction completes when the write operation has transferred over to the DMA bridge rather than wait until it reaches the actual peripheral. Therefore, the core can continue to execute other reads/writes without knowledge of the completion of the previous write.

In both cases, the operations complete on the peripheral bus in strict order of issue on the system bus. However, for writes—in particular completion of the write (as seen by the processor)—this does not mean that the write operation has taken effect. To guarantee this write operation has taken effect before allowing the core to proceed with additional execution, a read to any peripheral bus location is required (once the read operation has completed, all previous write operations have taken effect). Rather than be concerned about out-of-order sequences, developers using the PXA27x processor may want to disable the split-transaction feature.

3.0 Why Should Split Transaction be Enabled?

The split transaction feature should be enabled for two principal reasons: (1) to improve peripheral bus client accesses, and (2) to avoid system bus client underruns.

3.1 Improving Peripheral Bus Client Access

With the system bus as high as 208 MHz, the peripheral bus is always 26 MHz; the throughput of the system bus is eight times that of the peripheral bus. With split transaction, transaction requests across the peripheral bus can be posted to the DMA bridge, which makes parallel accesses to the peripheral bus space feasible.

3.2 Avoid System Bus Client Underun

When split transaction is disabled for read or write operations, the system bus is always owned by the XScale core until the read or write operation has completed, which causes underruns for system-bus clients such as LCD, camera, and others. Without split transaction enabled, high-resolution camera images (such as 1280x960x16bpp) may be corrupted.

4.0 How Split Transaction is Achieved

4.1 When Split Transaction is Disabled – Reads

A read transaction on the system bus is completed only after the DMA bridge receives the data from across the peripheral bus. There are no split responses, split completions, or retries in this mode.

See the following for a step-by-step description of a read transaction with split transaction disabled:

1. Core acquires the system bus
2. Core issues read command to the DMA bridge
3. Core stalls
4. Data is returned from the peripheral register to the DMA bridge, then to the core
5. Core activates and releases the system bus

4.2 When Split Transaction is Disabled – Writes

A write transaction on the system bus is completed only after the write data is sent across the peripheral bus. The targeted address location is guaranteed to be updated by the time the transaction completes on the system bus.

See the following for a step-by-step description of a write transaction with split transaction disabled:

1. Core acquires the system bus
2. Core issues write command and data to the DMA bridge
3. Core stalls
4. Data arrives at the peripheral register via the DMA bridge
5. DMA bridge indicates to the core that the write transaction is complete
6. Core activates and releases the system bus

4.3 When Split Transaction is Enabled – Reads

If the PIO transaction (PIO transaction is an I/O transaction read from a peripheral register or write to a peripheral register) is a read from a peripheral-address domain, the DMA split responds to the read instruction. The DMA bridge releases the system bus, then uses micro-coded instructions to read data from the peripheral bus. Once the read completes across the peripheral bus, the DMA controller completes the split transaction by re-capturing the system bus and completing the PIO read transaction. The core is stalled until the read is returned because this is programmed I/O. Any PIO transactions (reads or writes) that occur while the current PIO read transaction is between the split response and the split completion will be retried.

See the following for a step-by-step description for read transactions with split transaction enabled:

1. Core acquires the system bus

2. Core issues a read command to the DMA bridge
3. Core releases system bus and stalls
4. Data is returned from peripheral register to the DMA bridge
5. DMA bridge acquires the system bus
6. DMA bridge returns the data back to core
7. Core activates and releases the system bus

4.4 When Split Transaction is Enabled – Writes

If the programmed I/O transaction is a write instruction to a peripheral address domain, the DMA posts the write instruction. The DMA bridge indicates to the system bus that the PIO write is complete and then releases the system bus. The actual write transaction is then sent across the peripheral bus using micro-coded instructions. Any PIO instructions (reads or writes) between the time the write gets posted on the system bus and when the actual write completes cross the peripheral bus are retried.

See the following for the step-by-step description for write transactions with split transaction enabled:

1. Core acquires the system bus
2. Core issues write command and data to the DMA bridge
3. DMA bridge notifies the core that the write has completed
4. Core releases the system bus
5. Data arrives at the peripheral register via the DMA bridge

5.0 System Impact

5.1 System Bus Impact

Following is the system-bus impact with the split-transaction feature disabled/enabled.

- Read
 - If disabled, the system bus is released only when data returns to the core (that is, the core and system bus are stalled).
 - If enabled, the system bus is released once the read command reaches the DMA bridge (that is, the core is stalled but the system bus is not stalled).
- Write
 - If disabled, the system bus is released only when data reaches the peripheral bus space, if disabled (that is, the core and system bus are both stalled).
 - If enabled, the system bus is released once the write command and data reach the DMA bridge (that is core and system bus are not stalled).
- Summary

- Peripherals may see underrun if the split transaction feature is disabled.
- System bus can be shared more efficiently between multiple clients (USB host, LCD, core, camera, internal SRAM, DMA and memory controller), if enabled.

5.2 Core Execution Impact

Following is the core-execution impact with the split-transaction feature disabled/enabled.

- Read
 - Core continues executing only after data returns to the core, whether enabled or disabled.
- Write
 - If disabled, core continues executing only after data reaches the peripheral bus space.
 - If enabled, core continues executing once the command and data reach the DMA bridge.
- Summary
 - Read/write always stalls the core until complete.
 - Read always stalls the core until the data returns to core. The core can continue executing instructions without waiting for the write to complete.

6.0 How to Use Split Transaction

6.1 Ensure Earlier Write Takes Effect

As previously stated, when split transaction is enabled, the write is not safe—the core may continue execution assuming the previous write to the peripheral space took effect, when in fact it did not.

To ensure the earlier write to the peripheral-bus space takes effect, read back from the same address. For write-only registers, read from any location in the *same* peripheral address range. If the driver writes a few registers at one time, just one readback from the peripheral bus space is enough to ensure all the previous writes take effect.

Note: Usually a readback from any location in the same peripheral address range is enough to ensure that all of the previous writes take effect except for peripherals whose write is fairly slow (RTC and keypad, for example). In such cases, readback from the same address is sufficient.

6.2 Avoid Side Effects During Development

- Disable split transaction first when debugging a BSP

The default reset value for split transaction is “enabled.” To disable this feature, do so after each reset entry (power-on reset entry, wakeup-reset entry, etc.).

Enable split transaction to debug camera in high resolution.

- Enable split transaction when BSP is stable to gain performance

- Carefully check driver code for side effects of writing to the peripheral space

Examine any code that operates with the assumption that a peripheral-bus write has completed before executing the next line of code.

It is not necessary to read back after each write to achieve maximum parallelism. Analyze writes to the peripheral bus on a case-by-case basis.

- Disable the split-transaction feature when problems are encountered.

6.3 Write-back Sample Code

ASM code:

```

; r0 is the base address of the memory controller
; xlli_MSC2_value is the new value to set into MSC2
; xlli_MSC2_offset is the offset from memory controller to MSC2

ldr    r1, =xlli_MSC2_value          ; Get MSC2 setting
str    r1, [r0, #xlli_MSC2_offset]  ; Write the value out
ldr    r1, [r0, #xlli_MSC2_offset]  ; Read back to ensure earlier write takes
                                     effect

```

C code:

```

volatile unsigned_int32 *ptr=(volatile unsigned_int32*)0x48000010;
unsigned_int32 dummy;
*ptr=MSC2_VALUE;
dummy=*ptr;

```

7.0 Vulnerable Areas

7.1 Interrupt Controller

Some registers are accessible via the co-processor and peripheral bus; some are accessible only through the peripheral bus. Always use the same interface to access the same registers.

7.2 GPIO

A typical assumption during initialization is that the GPIO pins have been programmed for the appropriate functionality (which includes configuring GPIO pins for their alternate functionality, if required) as well as levels/directions set before the peripheral starts up. Because the GPIO controller is a peripheral-bus peripheral (due to the split-transaction feature), the earlier write to the GPIO registers may not take effect while the relevant peripherals have started. For example, when configuring the LCD controller, the driver must first configure the GPIO pins for LCD

functionality. Once the associated GPIOs are configured, the driver enables the LCD controller as soon as the write to the GPIO operation has returned—even though the GPIO operation may not have completed. System-bus clients (USBH, LCD, CI, etc.) are more likely to start up earlier relative to their associated GPIO pin state.

7.3 7.3 DMA

When peripheral-bus clients operate in DMA mode, they must operate their own registers and the DMA registers. Because the DMA is a system-bus client, the previous peripheral-register write may still not take effect while the write to the DMA register has started.

7.4 7.4 Interrupt Handler

Ensure that the register operation is correct within the interrupt handler; otherwise, it may be vulnerable to improper operation.

8.0 Sample Code to Disable Split Transaction

DMA Programmed IO Control Status Register (DPCSR)

Physical Address: 0x4000_00a4

This register is used for activating and monitoring posted writes and split reads, when software uses programmed I/O instructions to access the peripheral address domain via the DMA bridge.

DPCSR[BrgSplit]

Setting DPCSR[BrgSplit] to 1 activates split behavior.

Note: This control bit must be modified only when DPCSR[BrgBusy] is clear (no pending peripheral PIO transactions). Modifying this control bit when a PIO transaction is still pending might lead to unpredictable results and is, therefore, not recommended.

8.1 ARM format

```

1      ldr r0, =0x400000a4      ;physical address of DPCSR
      ldr r2, [r0]              ;judge busy bit
      ands r2, r2, #1
      bne b1
      str r2, [r0]              ;clear brgsplit bit

```

8.2 GNU format

```

1:      ldr r0, =0x400000a4      /* physical address of DPCSR */
      ldr r2, [r0]              /* judge busy bit */
      ands r2, r2, #1

```



Split Transaction Feature for the Intel® PXA27x Processor Family

```
bne 1b  
str r2, [r0]          /* clear brgsplit bit */
```

