CoreLink™
DMA Controller (DMA-330)
Errata Notice

This document contains all errata known at the date of issue in releases up to and including revision r0p0 of PL330 AXI DMA Controller and revision r1p1 of DMA-330 CoreLink DMA Controller.

Proprietary notice

Document confidentiality status

This document is Non Confidential.

Web address

http://www.arm.com/

Feedback on the product

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 document

If you have any comments on about this document, please send email to mailto:errata@arm.com giving:

- The document title
- The documents number
- The page number(s) to which your comments refer
- A concise explanation of your comments

General suggestion for additions and improvements are also welcome.

Contents

Introduction

Scope

This document describes errata categorised by level of severity. Each description includes:

- the current status of the defect
- where the implementation deviates from the specification and the conditions under which erroneous behavior occurs
- the implications of the erratum with respect to typical applications
- the application and limitations of a 'work-around' where possible

Categorisation of Errata

Errata recorded in this document are split into three levels of severity:

Category 1	Behavior that is impossible to work around and that severely restricts the use of the product in all, or the majority of applications, rendering the device unusable.
Category 2	Behavior that contravenes the specified behavior and that might limit or severely impair the intended use of specified features, but does not render the product unusable in all or the majority of applications.
Category 3	Behavior that was not the originally intended behavior but should not cause any problems in applications.

Change Control

23 Jul 2010: Changes in Document v9

Page	Status	ID	Cat	Summary
28	Updated	735717	Cat 2	Unaligned transfers may be corrupted

13 Jul 2010: Changes in Document v8

Page	Status	ID	Cat	Summary
28	New	735717	Cat 2	Unaligned transfers may be corrupted

20 Nov 2009: Changes in Document v7

Page	Status	ID	Cat	Summary
26	New	726945	Cat 2	A non-secure channel can potentially execute instructions that specify a secure peripheral interface
36	New	723164	Cat 3	Clearing an interrupt simultaneously with execution of DMAWFE causes only the event to be cleared
37	New	723172	Cat 3	DMAKILL in the same cycle as the last beat of a linefill does not completely clear the cache
38	New	723218	Cat 3	Aborting a channel during memory reset routine can cause incorrect reservation of FIFO entries
45	New	720043	Doc	DII 0193A DMAC (PL330) r0p0 IM - Clock enable functionality in pl330_pl080_interface is not documented
46	New	723170	Doc	Setting the endian swap size to be greater than the total burst size is not supported

17 Jul 2009: Changes in Document v5

Page	Status	ID	Cat	Summary
24	New	719340	Cat 2	Killing a channel which is in the "Waiting for peripheral" state does not clear the internal request state

22 May 2009: Changes in Document v4

Page	Status	ID	Cat	Summary
14	Updated	638719	Cat 2	Watchdog can incorrectly abort a channel when it is not locked up
18	Updated	658419	Cat 2	A non-secure channel can generate secure AXI accesses if the channel was previously secure and the CCR is not reset
20	Updated	680017	Cat 2	Running programs on multiple channels can cause unexpected lock-ups
22	New	716336	Cat 2	Destination address not aligned to the AXI bus width can cause deadlock due to incorrect reservation of MFIFO entries

27 Feb 2009: Changes in Document v3

Page	Status	ID	Cat	Summary
14	New	638719	Cat 2	Watchdog can incorrectly abort a channel when it is not locked up
18	New	658419	Cat 2	A non-secure channel can generate secure AXI accesses if the channel was previously secure and the CCR is not reset

19	New	673669	Cat 2	Instruction fetches are always marked as secure
20	New	680017	Cat 2	Running programs on multiple channels can cause unexpected lock-ups
31	New	537215	Cat 3	Dual-port RAMs and compiled register files can only be used if they accept simultaneous read/write to the same address
32	New	583615	Cat 3	Explicit perl paths causing run time issues on certain machines when configuring PL330
33	New	619517	Cat 3	Channel reuse with unaligned destination address may result in eventual MFIFO lock-up
35	New	638867	Cat 3	dmaasm incorrectly requires a prefix before immediate values on certain instructions
39	New	549180	Doc	DMAMOV CCR assembler syntax incorrect for SS, DS fields
40	New	587824	Doc	Missing information for "DMAWFP periph" and "DMALPEND" instructions
42	New	587830	Doc	Documentation is missing for dmaasm, dmalink and dmabuild
44	New	642067	Doc	Incorrect description of multi-channel load-store operation

09 Jan 2008: Changes in Document v2

Page	Status	ID	Cat	Summary
12	New	490518	Cat 2	A channel which has been used for a non-secure program cannot be re-used for a secure program

20 Dec 2007: Changes in Document v1

No changes in this document revision

Errata Summary Table

The errata associated with this product affect product versions as below.

A cell shown thus **X** indicates that the defect affects the revision shown at the top of that column.

ID	Cat	Summary of Erratum	r0p0-00rel0	r1p0-00rel0	r1p1-00rel0
549180	Doc	DDI 0424A - DMAC (PL330) r0p0 TRM - DMAMOV CCR assembler syntax incorrect for SS, DS fields	X		
587824	Doc	DDI 0424A - DMAC (PL330) r0p0 TRM - Missing information for "DMAWFP periph" and "DMALPEND" instructions	X		
587830	Doc	Documentation is missing for dmaasm, dmalink and dmabuild	X		
642067	Doc	DDI 0424A DMAC (PL330) r0p0 TRM - Incorrect description of multi-channel load-store operation	X		
720043	Doc	DII 0193A DMAC (PL330) r0p0 IM - Clock enable functionality in pl330_pl080_interface is not documented	X		
723170	Doc	Setting the endian swap size to be greater than the total burst size is not supported	X		
490518	Cat 2	A channel which has been used for a non-secure program cannot be re-used for a secure program	X		
638719	Cat 2	Watchdog can incorrectly abort a channel when it is not locked up	X		
658419	Cat 2	A non-secure channel can generate secure AXI accesses if the channel was previously secure and the CCR is not reset	X		
673669	Cat 2	Instruction fetches are always marked as secure	X		
680017	Cat 2	Running programs on multiple channels can cause unexpected lock-ups	X		
716336	Cat 2	Destination address not aligned to the AXI bus width can cause deadlock due to incorrect reservation of MFIFO entries	X		
719340	Cat 2	Killing a channel which is in the "Waiting for peripheral" state does not clear the internal request state	X		
726945	Cat 2	A non-secure channel can potentially execute instructions that specify a secure peripheral interface	X		
735717	Cat 2	Unaligned transfers may be corrupted	X	X	
537215	Cat 3	Dual-port RAMs and compiled register files can only be used if they accept simultaneous read/write to the same address	X		

ID	Cat	Summary of Erratum	r0p0-00rel0	r1p0-00rel0	r1p1-00rel0
583615	Cat 3	Explicit perl paths causing run time issues on certain machines when configuring PL330	X		
619517	Cat 3	Channel reuse with unaligned destination address may result in eventual MFIFO lock-up	X		
638867	Cat 3	dmaasm incorrectly requires a prefix before immediate values on certain instructions	X		
723164	Cat 3	Clearing an interrupt simultaneously with execution of DMAWFE causes only the event to be cleared	X		
723172	Cat 3	DMAKILL in the same cycle as the last beat of a linefill does not completely clear the cache	X		
723218	Cat 3	Aborting a channel during memory reset routine can cause incorrect reservation of FIFO entries	X		

Errata - Category 1

There are no Errata in this Category

Errata - Category 2

490518: A channel which has been used for a non-secure program cannot be re-used for a secure program

Status

Affects:	product DMA-330 AXI DMA Controller .
Fault status:	Cat 2, Present in: r0p0-00rel0, Fixed in r1p0-00rel0.

Description

The secure status of each channel within PL330 is stored in the CNS ("Channel Non-Secure") bit of the Channel Status Register ("CSR"). At reset, the status is set to 0 (secure); the status can only be changed by use of a `DMAGO` instruction.

A bug in the enable term for the register means that the bit can be set to 1 (non-secure) by a `DMAGO` instruction, but can not be cleared. Therefore if a channel is used for a non-secure program (CNS=1) then it cannot later be re-used for a secure program (requiring CNS=0).

Implications

If you wish to use both secure and non-secure channel programs, you must ensure that you do not mix programs with different security levels in one channel.

For example, if you issue the command:

```
DMAGO C0, 0x1000, ns
```

to cause channel 0 to execute a non-secure program (from address `0x1000`) then the CNS bit for channel 0 will be set to 1 (non-secure). After the program has completed, if you try to re-use channel 0 for a secure program by issuing the command:

```
DMAGO C0, 0x4000
```

then the CNS bit for channel 0 will remain at 1 (non-secure). Any subsequent instruction which attempts to utilise a secure resource (e.g. access a secure interrupt or peripheral, or specify secure AXI accesses in the Channel Control Register) will cause the program to abort with a security error.

Workaround

You must ensure that a channel which has been used for a non-secure program is not re-used for a secure program (unless the PL330 is reset). The following software workarounds are possible:

1. Determine which PL330 channels will be used for secure programs and which channels will be used for non-secure programs. Ensure that your software driver enforces the decision. This will prevent a non-secure channel being re-used by a secure program.

2. Ensure that a channel is never used for a non-secure program after it has been used for a secure program. This scenario is possible for systems which, for example, perform only secure DMA operations during initialisation and subsequently only perform non-secure DMA operations during normal operation.

3. After a channel has been used by a non-secure program, if you wish to re-use it for a secure program, reset the PL330 by driving the **aresetn** pin low for at least one clock cycle. This requires an external user-defined method of controlling the PL330 reset pin, and requires your software driver to ensure that all PL330 activity has ceased before causing the reset.

638719: Watchdog can incorrectly abort a channel when it is not locked up

Status

Affects:	product DMA-330 AXI DMA Controller .
Fault status:	Cat 2, Present in: r0p0-00rel0, Fixed in r1p0-00rel0.

Description

PL330 contains a watchdog function which is intended to abort programs which are poorly written or which attempt to cause a denial-of-service to other programs by using all available resources. In certain cases, when running programs on multiple channels, it is possible for very high latency to cause the watchdog within PL330 to abort a channel needlessly. The watchdog is controlled by the following equation, and it aborts a channel if the equation is true for that channel for 1024 consecutive cycles:

```
(((ld_denied_lsq | st_denied_lsq) &
  (ld_denied_pipe | st_denied_pipe | r_barrier | w_barrier)) |
 (ld_denied_lsq & st_denied_lsq)) &
 no_active_trans
```

Key:

- ld_denied_lsq - a DMALD in the load queue cannot initiate an AXI transaction because there is insufficient space in the MFIFO
- st_denied_lsq - a DMAST in the store queue cannot initiate an AXI transaction because there is insufficient data in the MFIFO
- ld_denied_pipe - a DMALD instruction cannot be executed because the load queue is full
- st_denied_pipe - a DMAST instruction cannot be executed because the store queue is full
- r_barrier - the channel is waiting for a read barrier to complete
- w_barrier - the channel is waiting for a write barrier to complete
- no_active_trans - the channel has no active AXI transactions

There are two main sources of latency that can cause this issue:

1. High latency on AXI transactions
2. Channel starvation due to programs using different burst sizes

The following example program can be used to illustrate both issues:

```
DMALP
    DMALD
    DMAST
    DMAWMB
DMALPEND
```

High latency on AXI transactions

Consider the case where 8 channels are all running the example program. Each channel has been configured to perform 4-beat aligned bursts and the MFIFO is 32 entries deep, therefore each channel will always be able to reserve space in the MFIFO (that is, ld_denied_lsq will never be asserted). The load and store queues are each configured to be 8 deep, the read and write issuing capabilities are configured to be 4. Therefore it is possible for the number of DMALD or DMAST instructions in the queues to be greater than the maximum permitted number of AXI transactions, thus it is possible for no_active_trans to be asserted.

Note: The maximum permitted number of AXI transactions is the lower of the two following values:

- The issuing capability of PL330
- The acceptance capability of the AXI slave interface to which PL330 is attached

If the store queue contains a DMAST instruction from a channel that does not currently have any active AXI transactions (no_active_trans is asserted) and has not already loaded data into the MFIFO, then st_denied_lsq is asserted for that channel. If the channel has also executed its DMAWMB instruction then w_barrier is asserted for that channel. The watchdog equation is now true and will remain so until the channel's DMALD instruction, which is currently in the load queue, is able to initiate an AXI transaction. The delay until this instruction is processed is dependent upon the latency of the current active AXI transactions, which belong to other channels.

Channel starvation due to programs using different burst sizes

Consider the case of a PL330 configuration with a 4-deep MFIFO where eight channels are each running the example program. Channel 0 is programmed to perform loads/stores of 4 words, and all other channels (1 to 7) are programmed to perform loads/stores of 1 word. One of the upper channels executes a DMALD, reserves 1 word of space in the MFIFO, and executes the associated AXI read. Channel 0 executes its DMALD and DMAST instructions and places them in the queues. It also executes its DMAWMB instruction. At this point it is now impossible for a DMALD from channel 0 to initiate an AXI transaction because it is impossible to reserve 4 words in the MFIFO. If the accesses for channels 1 to 7 are performed quickly on the AXI interface, it is likely that they will each continue to reserve a single word at a time, preventing channel 0 from starting a read transaction. If this continues for greater than 1024 cycles, channel 0 will be aborted by the watchdog because ld_denied_lsq, st_denied_lsq, w_barrier and no_active_trans are all asserted for channel 0.

Implications

The watchdog within PL330 can abort channels incorrectly in systems where the following two conditions apply:

- Transactions on the AXI interface can experience very high latency
- The read or write issuing capability can be completely used up by a number of channels which is less than the total number of channels currently executing programs

Workaround

All issues described in this defect can be avoided if you only run programs on a single channel within PL330. Further workarounds for the specific causes are described below.

High latency on AXI transactions

AXI latency can cause the watchdog to abort a channel if ld_denied_lsq or st_denied_lsq are asserted for that channel for 1024 cycles.

ld_denied_lsq

Limit the number of active DMALD instructions per channel so that the total number of MFIFO entries required by all active transactions cannot exceed the size of the MFIFO. You can achieve this using DMAWMB. For example:

```
DMALP
    DMALD
    DMAST
    DMAWMB
DMALPEND
```

In this example the DMAWMB command ensures that only a single DMALD/DMAST pair can be active for the channel. If the MFIFO depth is M and the number of channels is C then the maximum number of entries E that each channel can require is determined by:

```
E = M / C
```

For example, if the MFIFO is 32 entries deep and there are 8 channels, then each channel can require 4 entries at most to ensure safety.

st_denied_lsq

Delay DMAST instructions until the required data has been loaded into the MFIFO. You can achieve this using DMARMB. For example:

```
DMALP
    DMALD
    DMARMB
    DMAST
DMALPEND
```

In this example the DMAST will not be executed until after the data has been loaded into the MFIFO, therefore st_denied_lsq cannot be asserted.

Combined solution

Use both DMARMB and DMAWMB, together with the calculation of the maximum number of entries per channel, to guarantee that a channel will never lock up due to latency. For example:

```
DMALP
    DMALD
    DMARMB
    DMAST
    DMAWMB
DMALPEND
```

Channel starvation due to programs using different burst sizes

Asymmetric programs can cause the watchdog to abort a channel when a DMALD in the read queue is delayed because smaller transactions from other channels mean that there is never enough space in the MFIFO to process the DMALD into an AXI transaction.

To avoid this you can use the same mechanism as described for ld_denied_lsq (above) by inserting DMAWMB instructions to limit the number of active transactions on each channel. You must ensure that the sum of the number of entries requested by each channel is not greater than the size of the MFIFO.

658419: A non-secure channel can generate secure AXI accesses if the channel was previously secure and the CCR is not reset

Status

Affects:	product DMA-330 AXI DMA Controller .
Fault status:	Cat 2, Present in: r0p0-00rel0, Fixed in r1p0-00rel0.

Description

A non-secure channel can be initialised using the "DMAGO <ch>, <pc>, ns" command. Non-secure channels must not be able to generate secure AXI accesses, i.e. access with AWPROT[1]=0 or ARPROT[1]=0. The values placed on the AWPROT and ARPROT are set via the Channel Control Register (CCR).

It is not possible for a non-secure program to set the CCR such that secure AXI accesses are specified. However if the channel was previously used for a secure program then the previous CCR value (from that secure program) will still be set. If the channel has not previously been used then the CCR will still be at its reset value of 0x0 (implying secure source and destination accesses). In either of these cases, if the non-secure program does not attempt to change the CCR value, it will be able to perform secure accesses.

Implications

If you use a channel to execute a secure program which generates secure AXI accesses and then at the end of the program you do not change the CCR value to indicate non-secure accesses, a subsequent non-secure program using the channel could potentially generate secure AXI accesses.

If the first program that you execute on a channel after reset is a non-secure program and it does not change the CCR value then it will be able to generate secure AXI accesses.

Workaround

Before executing a non-secure program on any channel that has not been used since reset you must ensure that you change the value of the CCR for that channel to prevent the program from performing secure accesses. Execute the following program on every channel after reset:

```
DMAMOV CCR, SP2 DP2
DMAEND
```

At the end of any secure program you must set the CCR to indicate non-secure AXI accesses. Add the following instruction to the end of every secure program:

```
DMAMOV CCR, SP2 DP2
```

This will ensure that if the channel is re-used for a non-secure program it will not be able to perform secure accesses.

673669: Instruction fetches are always marked as secure

Status

Affects:	product DMA-330 AXI DMA Controller .
Fault status:	Cat 2, Present in: r0p0-00rel0, Fixed in r1p0-00rel0.

Description

Instruction fetches from PL330 are supposed to be performed with the same secure status as the thread for which the instructions are being fetched. For example, if the manager thread is secure (i.e. the DNS bit is 0) then all instruction fetches for the manager thread should also be secure; if a channel thread is non-secure (i.e. the channel CNS bit is 1) then all instruction fetches for that channel should be non-secure.

PL330 incorrectly performs all instruction fetches as secure, irrespective of the secure status of the threads.

Implications

Non-secure programs generate secure instruction fetches which could potentially be used to read from secure memory regions.

The only way to set the program counter (PC) for a channel is via the DMAGO instruction. Therefore there are only two ways in which a non-secure program can fetch from a secure address:

1. The non-secure thread is initiated with a PC value that points directly to a secure address. This requires the secure kernel to issue a DMAGO instruction with a secure PC address for a non-secure channel.
2. The non-secure thread is initiated with a PC value pointing to a program that is just below the start of a secure region. When the PC increments beyond the end of the non-secure address region it will start to read data from the secure region.

Note that a channel performing in this way will abort as soon as it tries to execute a value that is not a valid PL330 instruction from the secure region. However it is possible that some knowledge of the first few bytes might be derived from observing the operation of the non-secure channel.

Workaround

If you require PL330 to perform secure instruction fetches and you need to prevent non-secure channels from reading from a secure region, ensure that the following conditions are both met:

1. Ensure that the secure PL330 driver can not issue a DMAGO to a non-secure channel with an initial PC value that points to a secure region
2. Ensure that the memory region in which non-secure PL330 programs are placed is just below a region which always returns a data value of 0 (which will translate as DMAEND) or a region which always returns an error, thus causing the non-secure channel to abort

680017: Running programs on multiple channels can cause unexpected lock-ups

Status

Affects:	product DMA-330 AXI DMA Controller .
Fault status:	Cat 2, Present in: r0p0-00rel0, Fixed in r1p0-00rel0.

Description

Section 2.11.4 of the PL330 Technical Reference Manual (ARM DDI 0424A) describes how to avoid lock-ups due to the interaction of programs running on multiple channels. The equations given in figure 2-13 and figure 2-15 define the relationship between the maximum number of DMALD/DMAST instructions, the amount of data transferred by each of those instructions, the depth of the MFIFO, and the depth of the read/write queues.

The worst case example does not match the examples given in the text. Consider the following simple program fragment:

```
DMALPFE
    DMALD
    DMAST
DMALPEND
```

When running on a single channel in isolation, this program never locks-up (assuming that the MFIFO is of sufficient size to store the data required for a single DMALD/DMAST pair). If the same program runs on multiple channels (for example, all 8) and the MFIFO is small (i.e. it cannot hold the data from a DMALD from every channel) then it is possible that the timing of your system could cause the internal arbitration within PL330 to select the channels in an order that causes a lock-up.

For example, assume that each DMALD/DMAST transfers 4 words of data, that the MFIFO is only 4 words deep, and that the read and write queues are both 8 entries deep. This appears to satisfy the equation given in figure 2-13 because the apparent maximum number of DMALD instructions (one for each channel therefore eight in total) is less than the total of the read queue depth (8) plus the MFIFO depth divided by the burst size (4 / 4 = 1) - total 9.

Channel 0 executes a DMALD that fills the MFIFO and is then removed from the load queue. Each of the remaining channels then executes a DMALD so there are now 7 entries in the load queue. For some reason (for example a cache linefill), the round-robin channel arbitration policy in PL330 skips channel 0, and the remaining channels execute a DMAST each. There are now 7 entries in the write queue also. Channel 1 is selected next and issues another DMALD, filling the load queue. At this point there is only 1 entry left in the store queue, and only channels 0 and 1 can issue a DMALD. If channel 1 is selected again, then both the load and store queues are now filled with transactions that cannot complete, and channel 0 is unable to execute the one instruction (its DMAST) that can complete therefore the watchdog aborts the channel.

Implications

The description given in the TRM regarding lock-up avoidance is incomplete and does not fully describe the correct method to avoid watchdog aborts when using multiple channels. If you write programs purely according to the rules as described you might experience unexpected watchdog aborts.

Workaround

The example shows how arbitration effects can cause PL330 to select a channel several times in succession which can lead to a greater than expected number of active DMALD or DMAST transactions. The maximum number of active transactions that a channel can issue can be limited using the DMAWMB (write memory barrier) instruction. The example program fragment given above can be altered as follows:

```
DMALPFE
    DMAWMB
    DMALD
    DMAST
DMALPEND
```

The DMAWMB command causes the program to halt until all active DMAST instructions have completed - i.e. the AXI write transaction generated by the command has received a write response. Therefore, this program can now have at most 1 active DMALD and 1 active DMAST.

Note: in the example given above the DMAWMB instruction has been placed at the beginning of the loop to reduce latency. In this position, as soon as the last write transaction has completed the channel will execute the DMALD instruction next. If the DMAWMB is placed after the DMAST instruction, then when the last write transaction has completed the channel first has to execute the DMALPEND instruction before it can execute the DMALD.

With this method you can limit the maximum number of DMALD and DMAST transactions so that the equations given in figure 2-13 and figure 2-14 can be used safely.

When the programs for each DMA channel are created independently, for example by different software threads, a simpler but more restrictive pair of equations can be used. These specify a simple restriction on the number of active DMALD or DMAST instructions per channel based upon the configured queue depths and the number of channels in use:

```
Max. active DMALDs = Read Queue Depth / Number of active Channels
Max. active DMASTs = Write Queue Depth / Number of active Channels
```

In some cases, for example when specifying programming restrictions for end users of a system which contains PL330, the simpler equations may prove more practical.

716336: Destination address not aligned to the AXI bus width can cause deadlock due to incorrect reservation of MFIFO entries

Status

Affects:	product DMA-330 AXI DMA Controller .
Fault status:	Cat 2, Present in: r0p0-00rel0, Fixed in r1p0-00rel0.

Description

PL330 uses a mechanism to reserve space (measured in bytes) in the MFIFO before issuing the AXI read transaction resulting from a DMALD instruction. If the destination address is unaligned with respect to the AXI bus width then the initial unused bytes are included in the reservation made for the first DMALD. A timing issue within the logic means that if further DMALD instructions from the same channel are placed into the read queue before the initial reservation is complete, the latter DMALD instructions also (incorrectly) reserve the extra bytes due to the initial unalignment. These excess reservations can build up over successive iterations of the program until the MFIFO is apparently fully reserved, without actually containing any data.

Implications

Excess reservations in the MFIFO reduce the effective size of the MFIFO. This can cause performance degradation as channels compete for fewer available resources, or deadlock if the number of available MFIFO entries is fewer than that required for a program.

Workaround

If you specify a program with an unaligned destination address then you must insert a DMARMB (read memory barrier) instruction between the first and second DMALD instructions. For example, consider the following program:

```
# Specify 32-bit 4-beat bursts for source and destination
DMAMOV CCR, SS32 SB4 DS32 DB4
# Set source address.  The alignment of the source address does
# not affect this defect
DMAMOV SAR, 0x10000000
# Set destination address to an unaligned value (i.e. not aligned
# to the 32-bit transaction size)
DMAMOV DAR, 0x8003
# Perform bulk data move
DMALP 16
    DMALD
    DMAST
DMALPEND
# Re-program the CCR to allow the remaining 3 bytes to be stored
DMAMOV CCR, SS32 SB4 DS8 DB3
# Store the remaining data
DMAST
DMAEND
```

The defect will manifest if the second DMALD instruction is executed before the first DMALD instruction has been fully accepted by the read queue. To prevent this, you must insert a read barrier before the second DMALD instruction. Therefore the program shown above must be re-written as follows:

```
# Specify 32-bit 4-beat bursts for source and destination
DMAMOV CCR, SS32 SB4 DS32 DB4
# Set source address.  The alignment of the source address does
# not affect this defect
DMAMOV SAR, 0x10000000
# Set destination address to an unaligned value (i.e. not aligned
# to the 32-bit transaction size)
DMAMOV DAR, 0x8003
# Perform initial load/store pair
DMALD
DMAST
# Insert DMARMB to avoid defect 716336
DMARMB
# Perform bulk data move
DMALP 15
    DMALD
    DMAST
DMALPEND
# Re-program the CCR to allow the remaining 3 bytes to be stored
DMAMOV CCR, SS32 SB4 DS8 DB3
# Store the remaining data
DMAST
DMAEND
```

719340: Killing a channel which is in the "Waiting for peripheral" state does not clear the internal request state

Status

Affects:	product DMA-330 AXI DMA Controller .
Fault status:	Cat 2, Present in: r0p0-00rel0, Fixed in r1p0-00rel0.

Description

When a channel executes a DMAWFP instruction, the channel is placed into the "Waiting for peripheral" state and an internal signal named pipeline_req_active is set to indicate which peripheral is being waited for. When a peripheral request is received by PL330, if there is space then it is placed in the request receive buffer. If the matching pipeline_req_active is set then the received request is popped from the buffer and the channel is placed into the "Executing" state so that it can continue normal operation.

If the channel is terminated by the insertion of a DMAKILL instruction via the debug command registers whilst it is in the "Waiting for peripheral" state (i.e. before a matching request has been received) then the pipeline_req_active signal is not cleared. If the channel is then re-used for a later program it is possible for the request status of the peripheral and PL330 to become out of step, which can lead to the channel being paused forever in the "Waiting for peripheral" state. An example of this is shown below.

Channel 0 is used to run two programs in series, each of which services requests from peripheral 0. Peripheral 0 is designed to issue only a single active request: it can not issue another until the first request has been serviced and acknowledged. The two programs are both of the form shown below:

```
DMAMOV SAR ...
DMAMOV DAR ...
DMAMOV CCR ...
DMAFLUSHP P0
DMALPFE
    DMAWFP P0 burst
    DMALD
    DMASTP
DMALPEND
DMAEND
```

1. Program A running on channel 0 executes a DMAWFP 0 instruction to wait for a request from peripheral 0
2. Channel 0 enters the "Waiting for peripheral" state and sets the pipeline_req_active signal
3. The driver terminates program A by sending a DMAKILL instruction to channel 0
4. The driver creates program B and issues a DMAGO C0 instruction to run program B on channel 0
5. Program B executes a DMAFLUSHP 0 instruction to clear the request status of peripheral 0 and the internal counters in PL330
6. Peripheral 0 acknowledges the flush request and then issues a peripheral request to PL330

7. The peripheral request is matched against the pipeline_req_active signal and is removed from the receive buffer. The pipeline_req_active signal is cleared
8. Program B executes a DMAWFP 0 instruction to wait for a request from peripheral 0
9. Channel 0 enters the "Waiting for peripheral" state and sets the pipeline_req_active signal

At this point program B is waiting for a request, but peripheral 0 has already sent its single active request and is waiting for it to be serviced. Therefore the channel remains in the "Waiting for peripheral" state until it is killed.

Implications

If you use DMAKILL to kill a program that is in the "Waiting for peripheral" state then a following program that uses the same peripheral may become stuck in the "Waiting for peripheral" state until killed. In this case the stuck program will not be aborted automatically by the internal watchdog within PL330.

Workaround

The defect only manifests if program 0 has been killed during the "Waiting for peripheral state, and the request from the peripheral arrives after program 1 has issued its flush request (from the DMAFLUSHP instruction) and before program 1 executes its DMAWFP instruction. To avoid the defect you must prevent the peripheral from making requests during this period - the precise method of doing this will depend upon your system components.

The following example shows how the defect can be avoided. Program B must be re-written so that it asserts an interrupt after the DMAFLUSHP instruction:

```
DMAMOV SAR ...
DMAMOV DAR ...
DMAMOV CCR ...
DMAFLUSHP P0
DMASEV E0
DMALPFE
    DMAWFP P0 burst
    DMALD
    DMASTP
DMALPEND
DMAEND
```

The software driver must follow the schedule described below to ensure correct operation. This assumes that a previous program may or may not have been killed.

1. The driver disables the peripheral to prevent it from issuing DMA requests
2. The driver creates program B and issues a DMAGO C0 instruction to run program B on channel 0
3. Program B executes a DMASEV instruction which generates an interrupt
4. The driver polls the status of channel 0 until it observes that it has entered the "Waiting for peripheral" state
5. The driver enables the peripheral to allow it to issue DMA requests

726945: A non-secure channel can potentially execute instructions that specify a secure peripheral interface

Status

Affects:	product DMA-330 AXI DMA Controller .
Fault status:	Cat 2, Present in: r0p0-00rel0, Fixed in r1p0-00rel0.

Description

The Trustzone security status of each peripheral interface in PL330 is set by the boot_periph_ns input bus. You can read the value of this input from Configuration Register 4 (see the PL330 Technical Reference Manual for details). The following instructions interact with a specified peripheral interface:

- DMAWFP
- DMALDP
- DMASTP
- DMAFLUSHP

If you try to execute one of these instructions from a non-secure channel, and the instruction specifies a secure peripheral, then the instruction should be aborted.

In PL330 r0p0, instead of checking the peripheral interface specified by the instruction, the DMAC checks the registered peripheral value for the channel executing that instruction. This registered peripheral value is set to 0 at reset, and is updated whenever a DMAWFE instruction is successfully executed by that channel.

Therefore, if the security status of peripheral 0 is set to non-secure then a non-secure channel can execute any of the peripheral-related instructions listed above and specify any peripheral interface, until the registered peripheral value is changed to indicate a secure peripheral. If the security status of peripheral 0 is set to secure then a non-secure channel will be prevented from executing any of the peripheral-related instructions listed above until the registered peripheral value is changed to indicate a non-secure peripheral.

NOTE: Although it is possible for a non-secure channel to execute a DMALDP or DMASTP instruction with a secure peripheral interface specified, it is not possible for the non-secure channel to perform secure AXI accesses.

Implications

If a non-secure channel executes a DMAWFP instruction that specifies a secure peripheral, then one or more requests from that peripheral might be accepted and discarded by PL330 before the channel that is correctly programmed to accept those responses can receive them. This might cause the correct channel to wait indefinitely because the peripheral might send no more requests.

If a non-secure channel executes a DMALDP, DMASTP or DMAFLUSHP instruction that specifies a secure peripheral, then an acknowledge or flush command will be sent to the specified peripheral. This will cause the peripheral to send new requests, which can cause the channel that is correctly programmed to accept those responses to perform excess bus transfers and therefore lose data or transfer invalid data.

Workaround

To prevent a non-secure program from accessing a secure peripheral interface you must run the following program on every channel after reset, where "Pn" specifies a secure peripheral:

```
DMAWFP Pn
DMAEND
```

This will cause the registered peripheral value for each channel to be updated to indicate the specified secure peripheral.

735717: Unaligned transfers may be corrupted

Status

Affects:	product DMA-330 AXI DMA Controller .
Fault status:	Cat 2, Present in: r0p0-00rel0,r1p0-00rel0, Fixed in r1p1-00rel0.

Description

For a configuration with more than one channel, if any of channels 1 to 7 is performing transfers between certain types of misaligned source and destination addresses, then the output data may be corrupted by the action of channel 0.

Data corruption might occur if all of the following are true:

1. Two beats of AXI read data are received for one of channels 1 to 7
2. Source and destination address alignments mean that each read data beat is split across two lines in the data buffer (see Splitting data, below)
3. There is one idle cycle between the two read data beats
4. Channel 0 performs an operation that updates channel control information during this idle cycle (see Updates to channel control information, below)

Splitting data

Depending upon the programmed values for the DMA transfer, one beat of read data from the AXI interface may need to be split across two lines in the internal data buffer. This occurs when the read data beat contains data bytes which will be written to addresses that wrap around at the AXI interface data width, so that these bytes could not be transferred by a single AXI write data beat of the full interface width.

Most applications of DMA-330 do not split data in this way, so are NOT vulnerable to data corruption from this defect.

The following cases are **NOT** vulnerable to data corruption because they do not split data:

- Byte lane offset between source and destination addresses is 0

When source and destination addresses have the same byte lane alignment, the offset is 0 and a wrap operation that splits data cannot occur.

- Byte lane offset between source and destination addresses is a multiple of source size

Source size in CCRn	Allowed offset between SARn and DARn
SS8	any offset allowed.
SS16	0,2,4,6,8,10,12,14
SS32	0,4,8,12
SS64	0,8
SS128	0

The following case might split data if there is a byte lane offset between source and data but is **NOT** vulnerable to data corruption because any splitting of data is treated by a different mechanism within the DMAC:

- Destination has a fixed address and a size less than the AXI bus width

For example, in a configuration with a 64-bit AXI interface, writing to an 8-, 16- or 32-bit peripheral FIFO at a fixed address.

The following program is an example that **MIGHT** be vulnerable to data corruption. In a DMAC configured with a 64-bit AXI interface:

```
DMAMOV SAR 0x1004
DMAMOV DAR 0x2000    ;; source/dest offset is 4
DMAMOV CCR SB3 SS64 DB2 DS64
DMALD                ;; read 4 bytes from 0x1004-0x1007 + 8 bytes from 0x1008-0x100F + 8
bytes from 0x1010-0x1017
DMAST                ;; write 8 bytes to 0x2000-0x2007 + 8 bytes to 0x2008-0x200F
...
```

In this program:

Read data beat 1 reads 4 bytes from 0x1004-0x1007 which will be written to 0x2000-0x2003.

Read data beat 2 reads 8 bytes from 0x1008-0x100F and these will wrap from 0x2004-0x2007 to 0x2008-0x200B.

Read data beat 3 reads 8 bytes from 0x1010-0x1017 and these will wrap from 0x200C-0x200F to 0x2010-0x2013.

If this program runs on one of channels 1 to 7, then read data beat 3 is vulnerable to the data corruption under the conditions described for timing and channel 0.

Updates to channel control information

The data splitting operation is affected by certain channel control information.

The control information for channel 0 is updated when either of the following occur:

- Channel 0 executes the first `DMALD` instruction in the channel thread program.
- Channel 0 executes the first `DMALD` or `DMAST` instruction following a change to any of the Source Address Register, Destination Address Register or Channel Control Register where that change affects the byte lane offset between source and destination addresses.

A typical example of such a change is using a `DMAADDH` instruction to stride the addresses for transfers between sparse data and packed data.

For example, the following program gathers 4 bytes from 0x5000, 0x5008, 0x5010, 0x5018 and writes them to 0x6000-0x6003. The first `DMALD` instruction updates the channel control information because it is the first in the program. The subsequent three `DMALD` instructions within the loop all cause further updates to the channel control information because they are changing the offset between source and destination address to 1, 2 and 3 respectively.

```
DMAMOV SAR 0x5000
DMAMOV DAR 0x6000
DMAMOV CCR SB1 SS8 DB1 DS32
```

```
DMALD           ;; read from 0x5000 for destination 0x6000, offset 0
DMAADDH SAR 7
DMALD           ;; read from 0x5008 for destination 0x6001, offset 1
DMAADDH SAR 7
DMALD           ;; read from 0x5010 for destination 0x6002, offset 2
DMAADDH SAR 7
DMALD           ;; read from 0x5018 for destination 0x6003, offset 3
DMAST           ;; write 0x6000-0x6003
...
```

Implications

Incorrect data may be output for AXI data writes.

Workaround

The possible workarounds depend on how the DMAC is being used by the application.

Determine how many of the channel programs running concurrently will split read data across two lines of the internal data buffer (as described above under Splitting data).

(a) Only one channel program splits read data

If only one channel program splits read data then run this program on channel 0.

Non-splitting programs are not vulnerable to data corruption from this defect so can run concurrently on channels 1 to 7.

(b) More than one channel program splits read data

If more than one channel program splits read data and they must run concurrently then run these programs on any of channels 1 to 7, and do not run any program concurrently on channel 0.

Errata - Category 3

537215: Dual-port RAMs and compiled register files can only be used if they accept simultaneous read/write to the same address

Status

Affects:	product DMA-330 AXI DMA Controller .
Fault status:	Cat 3, Present in: r0p0-00rel0, Fixed in r1p0-00rel0.

Description

The main Verilog arrays within PL330 are designed so that you can replace them with dual-port RAM or compiled register file structures. However many dual-port RAMs and register file designs do not permit a simultaneous read and write to the same address. PL330 does not prevent such accesses and therefore can only use dual-port RAMs and compiled register files that do permit a simultaneous read and write to the same address.

Implications

You cannot use a dual-port RAM or compiled register file which does not support a simultaneous read and write to the same address as a replacement for a Verilog array in PL330. If you do, you may experience random failures due to unexpected program behaviour.

Workaround

None.

583615: Explicit perl paths causing run time issues on certain machines when configuring PL330

Status

Affects:	product DMA-330 AXI DMA Controller .
Fault status:	Cat 3, Present in: r0p0-00rel0, Fixed in r1p0-00rel0.

Description

Some of the Perl scripts supplied with PL330 call the Perl executable from a fixed location, rather than by searching for it on your path. The affected scripts are all in the shared/logical/pl330_dma/bin directory:

dat2bin.pl

dmaasm

dmabuild

dmalink

split128.pl

Each script calls Perl using the following first line in the script:

```
#!/usr/local/bin/perl -w
```

This does not work if you do not have access to a perl executable at this location.

Implications

If you try to use the PL330 scripts in an environment that does not have a Perl executable at the location specified, the scripts fail.

Workaround

Replace the first line of the affected scripts with the following routine so that your shell searches for the Perl executable on your path:

```
eval "exec perl -w -S $0 $@" # -*- Perl -*-
  if ($running_under_some_sh);
  undef ($running_under_some_sh);
```

619517: Channel reuse with unaligned destination address may result in eventual MFIFO lock-up

Status

Affects: product DMA-330 AXI DMA Controller .

Fault status: Cat 3, Present in: r0p0-00rel0, Fixed in r1p0-00rel0.

Description

The data merge buffer in PL330 ensures that MFIFO entries are used as efficiently as possible by determining when new data is contiguous to the data already stored in the open MFIFO entry. For example, a 16-bit load has been performed (on a 32-bit PL330) and the data has been stored in bits [15:0] of an MFIFO entry. The next 16-bit load needs to place data into bits [31:16]. This is contiguous and therefore is permitted by the merge buffer.

If the final data in a DMA sequence does not align with the upper boundary of an MFIFO entry the merge buffer determines that some contiguous data may arrive next. In this case the channel holds the MFIFO entry until data does arrive or the channel is stopped or killed.

An error in the MFIFO reservation logic means that when this final reserved entry is freed in the MFIFO when stopping a channel, the reservation counter is not cleared. When the channel is next used, if the new program sets up DAR and CCR values such that the next data can be contiguous to the last data from the previous program, then the reservation logic is still not cleared and therefore the number of free entries in the MFIFO appears to be artificially reduced from the point of view of that channel. Repetition of this sequence can result in the data merge buffer state indicating that all MFIFO entries have been reserved, which will lock-up the channel causing a channel timeout abort.

For example, channel 0 is called with the following program:

```
;; Set up 1-beat 8-bit bursts
DMAMOV   CCR, SB1 SS8 DB1 DS8
DMAMOV   SAR, 0x1000
DMAMOV   DAR, 0x2000
;; Loop around 5 load/store pairs
DMALP    5
    DMALD
    DMAST
DMALPEND
;; End the program
DMAEND
```

At the end of this program, the DAR holds the value 0x2005. Channel 0 is now called with this program:

```
;; Set up 1-beat 8-bit bursts
DMAMOV   CCR, SB1 SS8 DB1 DS8
DMAMOV   SAR, 0x4000
DMAMOV   DAR, 0x5005
;; Loop around 5 load/store pairs
DMALP    5
```

```
        DMALD
        DMAST
    DMALPEND
    ;; End the program
    DMAEND
```

Because the CCR values have not changed and the DAR value of the second program matches the alignment of the final DAR value at the end of the first program, the reservation logic is not cleared. Therefore the MFIFO entry reservation counter is now incorrect. If this repeats then the channel will eventually lock-up and abort.

Implications

A set of programs that exhibit this behaviour cause the affected channel or channels to lock up.

Workaround

Add the following instruction to the start of each program:

```
    DMAMOV   DAR, 0x0
```

This forces the reservation logic to clear the reservation counter before executing the rest of your program. The extra instruction has no effect on your program, assuming that you do set the DAR value correctly in the body of the program.

For example, the second program listed in the description section becomes:

```
    ;; Clear DAR to force reservation logic to be reset
    DMAMOV   DAR, 0x0
    ;; Set up 1-beat 8-bit bursts
    DMAMOV   CCR, SB1 SS8 DB1 DS8
    DMAMOV   SAR, 0x4000
    DMAMOV   DAR, 0x5005
    ;; Loop around 5 load/store pairs
    DMALP    5
        DMALD
        DMAST
    DMALPEND
    ;; End the program
    DMAEND
```

638867: dmaasm incorrectly requires a prefix before immediate values on certain instructions

Status

Affects:	product DMA-330 AXI DMA Controller .
Fault status:	Cat 3, Present in: r0p0-00rel0, Fixed in r1p0-00rel0.

Description

The assembler (dmaasm) provided with PL330 requires a prefix to identify numerical immediate values for the following instructions: DMASEV, DMAWFE, DMALDP, DMASTP, DMAFLUSHP, DMAWFP. The assembler requires a prefix "p" for peripheral numbers, and a prefix of "e" for event numbers.

For example, the following instructions cause the assembler to report an error:

```
DMASEV 0
DMALDP 7
```

The following instructions do not cause an error:

```
DMASEV E0
DMALDP P7
```

Implications

A PL330 program that is written correctly according to the TRM, that is without a prefix for the affected instructions, causes dmaasm to report an error when the program is compiled.

Workaround

When you write a PL330 program for compilation with dmaasm, you must specify a prefix for immediate values that specify a peripheral or event.

For an immediate value that specifies a peripheral you must use a "p" prefix, for example:

```
DMALDP P1
DMASTP P6
DMAFLUSHP P3
DMAWFP P2
```

For an immediate value that specifies an event you must use an "e" prefix, for example:

```
DMASEV E5
DMAWFE E1
```

723164: Clearing an interrupt simultaneously with execution of DMAWFE causes only the event to be cleared

Status

Affects:	product DMA-330 AXI DMA Controller .
Fault status:	Cat 3, Present in: r0p0-00rel0, Fixed in r1p0-00rel0.

Description

If you are using both events and interrupts then it is possible for a write to the interrupt clear register not to clear the specified interrupt. This will only occur if the write to the interrupt clear register occurs in the same cycle that a DMAWFE (wait for event) instruction is executed.

Implications

If your write to the interrupt clear register coincides with a DMAWFE instruction then the interrupt will not be cleared.

Workaround

If you are using both events and interrupts, your interrupt clear routine must check the interrupt status register after it has written to the interrupt clear register. If the interrupt is still set, the write to the interrupt clear register must be repeated.

723172: DMAKILL in the same cycle as the last beat of a linefill does not completely clear the cache

Status

Affects:	product DMA-330 AXI DMA Controller .
Fault status:	Cat 3, Present in: r0p0-00rel0, Fixed in r1p0-00rel0.

Description

If you insert a DMAKILL via the debug registers to terminate a channel, and the execution of that DMAKILL coincides with the last cycle of a linefill for that thread, then the linefill data is still placed in the cache and the cache_valid bit remains set when it should be cleared.

Implications

If the affected channel is re-used for a new program, and the address of the new program matches the address tagged in the cache from the previous program, then the cached instructions will be executed. If the instructions have been changed in memory then this implies that the wrong instructions will be executed by the new program.

Workaround

You can force a channel to flush its cache entries by running any program which executes a DMAEND instruction. Therefore, if you use DMAKILL to terminate a program, you must then execute a second program containing just the DMAEND instruction on the channel on which that first program was executing.

723218: Aborting a channel during memory reset routine can cause incorrect reservation of FIFO entries

Status

Affects:	product DMA-330 AXI DMA Controller .
Fault status:	Cat 3, Present in: r0p0-00rel0, Fixed in r1p0-00rel0.

Description

When you configure PL330 for memory implementation of the main FIFO arrays, the pointer array is synchronously reset by a routine that steps through each location writing the correct value before normal operation can begin. The initialisation routine lasts for a number of clock cycles equal to the configured depth of the FIFO, therefore the maximum length of the routine is 1024 clock cycles.

If a channel executes a DMALD instruction and the resulting AXI read transaction receives an error response before the reset routine is complete, then FIFO entries for that DMALD incorrectly remain reserved and cannot be re-used.

This defect is only likely to occur in the following circumstances:

- You have set PL330 to boot from reset (the boot_from_pc input is tied high), or a DMAGO instruction is inserted by another device immediately after reset
- The manager program and any channel programs are loaded from fast memory, so that a DMALD instruction can be executed before the memory reset routine is complete
- The memory region from which PL330 is loading programs can and does issue an AXI error response to PL330

Implications

Incorrect reservation of FIFO entries in this way will artificially reduce the available FIFO size for all other programs.

Workaround

If your PL330 configuration implements the FIFO as memories and you set up your system so that it can execute programs immediately, you must make sure that one of the following statements is true:

- The address regions which are accessed by these initial programs during the memory reset sequence can not issue an AXI error response; or:
- The initial programs do not execute a DMALD instruction until after the memory reset sequence has completed

Errata - Documentation

549180: DDI 0424A - DMAC (PL330) r0p0 TRM - DMAMOV CCR assembler syntax incorrect for SS, DS fields

Status

Affects:	product DMA-330 AXI DMA Controller .
Fault status:	Doc, Present in: r0p0-00rel0, Fixed in r1p0-00rel0.

Description

Section 4.4.6 "DMAMOV CCR" of the "PrimeCell DMA Controller (PL330) Technical Reference Manual" (ARM DDI 0424A) gives incorrect descriptions for the SS (source size) and DS (destination size) fields. SS and DS specify the size in bits not bytes, and allowable values are 8, 16, 32, 64, 128.

Implications

none

Workaround

none

587824: DDI 0424A - DMAC (PL330) r0p0 TRM - Missing information for "DMAWFP periph" and "DMALPEND" instructions

Status

Affects:	product DMA-330 AXI DMA Controller .
Fault status:	Doc, Present in: r0p0-00rel0, Fixed in r1p0-00rel0.

Description

The PL330 Technical Reference Manual (ARM DDI 0424A) does not give a complete description of the operation of the "DMAWFP periph" and "DMALPEND" instructions. An extra flag (called "request_last") exists for each channel within PL330 to track when a last request has been received from a peripheral, i.e. a peripheral DMA request with dr_last asserted.

The "DMAWFP periph" instruction (see section 4.3.19 on page 4-22) has the following additional behaviour: when a DMA request is received from the specified peripheral with dr_last asserted the request_last flag is set for the channel.

When you use DMALPEND (see section 4.3.8 on page 4-11) to indicate the end of an infinite loop, i.e. one which was started using the DMALPFE instruction, the following behaviour takes effect:

1. The nf bit is set to 0 in the DMALPEND instruction to indicate the end of an infinite loop
2. The lc bit is set to 1 in the DMALPEND instruction
3. The bs and x bits in the DMALPEND instruction are ignored and must be set to 0. Therefore you should only use DMALPEND to terminate an infinite loop, and not the DMALPENDS or DMALPENDB variants.
4. When the DMALPEND instruction (with nf=0) is executed, if the request_last flag for the channel is set then the DMALPEND instruction is treated as a DMANOP and the request_flag is cleared.

These behaviours allow you to specify an instruction loop to service all the DMA requests from a peripheral until the last request from that peripheral. For example:

```
;; Setup source and destination addresses
DMAMOV SAR, 0x100000
DMAMOV DAR, 0x400000
;; Setup 4-beat 32-bit bursts for source and destination
DMAMOV CCR, SB4 SS32 DB4 DS32
;; Infinite loop to deal with DMA requests from peripheral 4
DMALPFE
    DMAWFP P4, periph
    ;; If a burst request is received...
    DMALDPB P4
    DMASTB
    ;; If a single request is received...
    DMALDPS P4
    DMASTS
;; Jump back to the beginning of the loop unless dr_last was set on the request
DMALPEND
DMAEND
```

Implications

none

Workaround

none

587830: Documentation is missing for dmaasm, dmalink and dmabuild

Status

Affects:	product DMA-330 AXI DMA Controller .
Fault status:	Doc, Present in: r0p0-00rel0, Fixed in r1p0-00rel0.

Description

The PL330 documentation does not describe how to re-use the supplied assembler and linker utilities (dmaasm, dmalink, dmabuild). The following sections give an overview of their usage.

dmaasm

```
    dmaasm -o <output file> -i <input file> [-D <var1>=<value1> ... -D <varN>=<valueN>]
```

Assembles a PL330 assembler source file into an ASCII hex format for use by dmalink.

The -D option can be used to specify replacement values for variables within the source code. Variables are indicated by $ and are bounded by white-space. For example, if the source code contains the following line:

```
    DMAGO  C0,   $C0
```

Then the option -D C0=0xF000 will cause the code to be compiled as if written as follows:

```
    DMAGO  C0,   0xF000
```

dmalink

```
    dmalink -o <output file> -i <input file> -a <base address> [-mem] [-bw <bus_width>]
```

Links multiple ASCII output files from dmaasm into a single ASCII hex file.

By default, a file suitable for use with the RVML (ARM RealView Model Library) AXI memory model is created. When the -mem option is used a file suitable for use in a Verilog test environment (using $readmemh) is created.

By default a test environment memory width of 32-bits is assumed. When the -bw option is used the memory width can be specified: valid values are 32, 64, 128.

dmabuild

```
    dmabuild $prog -o <output file> -a <base address> -m <manager file> \
                        [-c0 <channel0 file> ... -c7 <channel7 file>] [-mem] \
                        [-bw <bus_width>] [-pad <cache_line_words>]
```

dmabuild allows you to compile a multi-channel test program comprising one program for the manager thread plus a single program for each of one or more channels. By using variable substitution you do not need to specify the base address of each channel program - these are calculated automatically and placed into the manager program. The manager program must use the correct pre-defined variables, e.g.:

```
    DMAGO  C0,   $C0
    DMAGO  C1,   $C1
    DMAGO  C2,   $C2
    DMAGO  C3,   $C3
    DMAGO  C4,   $C4
    DMAGO  C5,   $C5
    DMAGO  C6,   $C6
    DMAGO  C7,   $C7
    DMAEND
```

The -mem and -bw options operate exactly as for dmalink

By default all compiled programs are placed directly adjacent in memory. The -pad option allows you to specify that empty bytes are placed between the programs so that each program is aligned to the number of words in a cache line.

Implications

none

Workaround

none

642067: DDI 0424A DMAC (PL330) r0p0 TRM - Incorrect description of multi-channel load-store operation

Status

Affects:	product DMA-330 AXI DMA Controller .
Fault status:	Doc, Present in: r0p0-00rel0, Fixed in r1p0-00rel0.

Description

Figure 2-14 on page 2-41 of the PL330 Technical Reference Manual (ARM DDI 0424A) and the associated descriptive text on pages 2-40 and 2-41 are slightly incorrect. On page 2-40 the text states that:

"Figure 2-14 on page 2-41 shows a DMAC operating with four active DMA channels that are each executing similar code to that shown in Example 2-4 on page 2-38. Using a read queue depth of 7 and an MFIFO depth of 8, the DMAC can execute the program without a lock-up condition occurring."

This is incorrect because the code on page 2-38 indicates that the DMALD instruction will load 8 words of data and the DMAST instruction will store 16 words of data. This is not possible with a MFIFO depth of 8: attempting to load or store more data than can fit in the MFIFO is a programming error and will cause the channel to abort. The operation shown in figure 2-14 is only correct (for the given program) if the MFIFO depth is 16.

The following note at the bottom of page 2-41 is incorrect and should be ignored - PL330 cannot perform a partial store:

"In Figure 2-14, when the DMAC executes DMAST for DMA channel 0, it writes the destination data but it can only provide half of the intended data. This occurs because the DMAC could not complete the DMALD (2) as the MFIFO was full."

Implications

none

Workaround

none

720043: DII 0193A DMAC (PL330) r0p0 IM - Clock enable functionality in pl330_pl080_interface is not documented

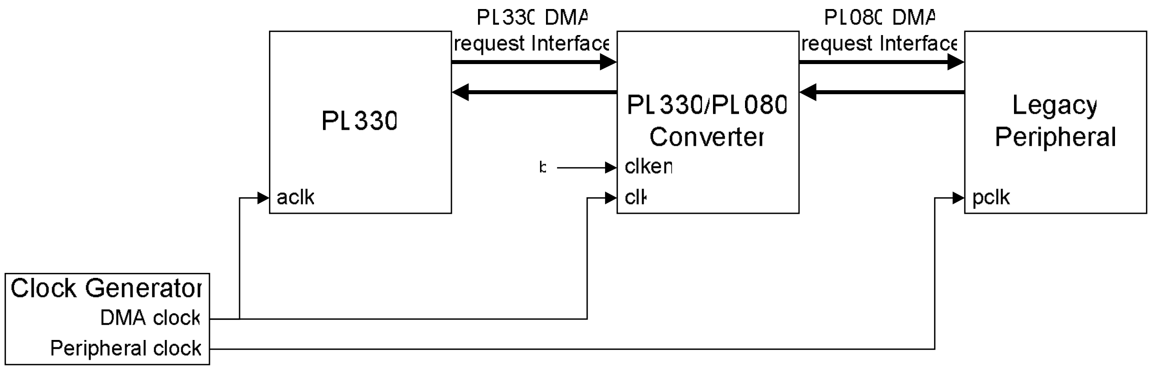
Status

Affects:	product DMA-330 AXI DMA Controller .
Fault status:	Doc, Present in: r0p0-00rel0, Fixed in r1p0-00rel0.

Description

Appendix A of the PL330 Integration Manual describes the DMA Request Interface Converter, which allows you to connect a legacy peripheral that implements a PL080-style DMA request interface to PL330. Figure A-1 "Converter connections" on p.A-2 illustrates the integration of the converter block into a system but does not include the clken input port of the converter.

The purpose of the clken input is to allow the entire converter to run at a frequency lower than that of the clock connected to the clk input. If clken is not tied high, then only the rising edges of clk which coincide with clken=1 are used to advance the internal state of the converter. The converter must operate at the same frequency as PL330 so that the DMA request and acknowledge signals between the converter and PL330 are interpreted correctly by both components. Therefore you should integrate the converter by connecting its clk input to the same system clock as that which is connected to the aclk input of PL330, and tying the clken input high as shown in the following diagram.



Note that the PL080 DMA request interface is an asynchronous interface. Therefore the integration method illustrated above will work correctly as long as the frequency of the Peripheral clock is less than or equal to the DMA clock.

Implications

none

Workaround

none

723170: Setting the endian swap size to be greater than the total burst size is not supported

Status

Affects:	product DMA-330 AXI DMA Controller .
Fault status:	Doc, Present in: r0p0-00rel0, Fixed in r1p0-00rel0.

Description

For transfers that use an incrementing destination address, you must program the CCRn Register so that dst_burst_len×dst_burst_size ≥ endian_swap_size. For example, if endian_swap_size = b010 (32-bit) and dst_burst_size = b001 (2 bytes per beat) then you can program dst_burst_len = b0001-b1111 (2-16 data transfers). See the description of the Channel Control Registers for details of these values.

Implications

none

Workaround

none

Errata – Driver Software

There are no Errata in this Category