TMS320DM6467
Digital Media System-on-Chip (DMSoC)
Silicon Revisions 3.0, 1.1, and 1.0

Silicon Errata

TEXAS INSTRUMENTS

Literature Number: SPRZ251G

September 2007–Revised October 2010

List of Figures

List of Tables

Silicon Errata

SPRZ251G–September 2007–Revised October 2010

TMS320DM6467 DMSoC Silicon Revisions 3.0, 1.1, and 1.0

1 Introduction

This document describes the known exceptions to the functional specifications for the TMS320DM6467 DMSoC devices (i.e., TMS320DM6467). For more detailed information on these devices, see the device-specific data manual:

- *TMS320DM6467 Digital Media System-on-Chip* data manual (Literature Number SPRS403)

Throughout this document, unless otherwise specified, TMS320DM646x and DM646x, refer to the TMS320DM6467 device. For additional peripheral information, see the latest version of the *TMS320DM646x DMSoC Peripherals Overview* Reference Guide (Literature Number SPRUEQ0).

The advisory numbers in this document are not sequential. Some advisory numbers have been moved to the next revision and others have been removed and documented in the user's guide. When items are moved or deleted, the remaining numbers remain the same and are not resequenced.

1.1 Device and Development-Support Tool Nomenclature

To designate the stages in the product development cycle, TI assigns prefixes to the part numbers of all DSP devices and support tools. Each DSP commercial family member has one of three prefixes: TMX, TMP, or TMS (e.g.,TMX320DM6467ZUT). Texas Instruments recommends two of three possible prefix designators for its support tools: TMDX and TMDS. These prefixes represent evolutionary stages of product development from engineering prototypes (TMX/TMDX) through fully qualified production devices/tools (TMS/TMDS).

Device development evolutionary flow:

TMX — Experimental device that is not necessarily representative of the final device's electrical specifications.

TMP — Final silicon die that conforms to the device's electrical specifications but has not completed quality and reliability verification.

TMS — Fully-qualified production device.

Support tool development evolutionary flow:

TMDX — Development-support product that has not yet completed Texas Instruments internal qualification testing.

TMDS — Fully qualified development-support product.

TMX and TMP devices and TMDX development-support tools are shipped against the following disclaimer:

"Developmental product is intended for internal evaluation purposes."

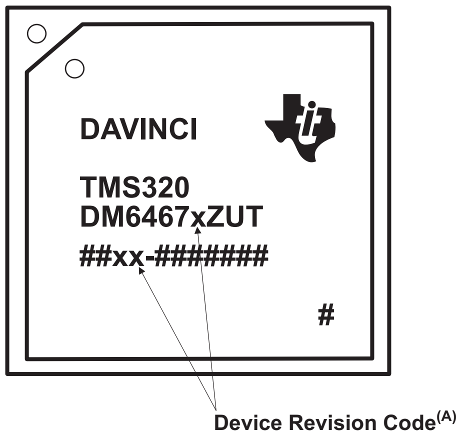
TMS devices and TMDS development-support tools have been characterized fully, and the quality and reliability of the device have been demonstrated fully. TI's standard warranty applies.

Predictions show that prototype devices (TMX or TMP) have a greater failure rate than the standard production devices. Texas Instruments recommends that these devices not be used in any production system because their expected end-use failure rate still is undefined. Only qualified production devices are to be used.

TI device nomenclature also includes a suffix with the device family name. This suffix indicates the package type (for example, ZUT), the temperature range (for example, "Blank" is the commercial temperature range), and the device speed range in megahertz (for example, "Blank" is the default [594-MHz DSP, 297-MHz ARM9]).

1.2 Revision Identification

Figure 1 provides an example(s) of the TMS320DM6467 device markings. The device revision can be determined by the symbols marked on the top of the package.



A. “#” denotes an alphanumeric character

Figure 1. Example, Device Revision Codes for TMS320DM6467 (ZUT Package)

Silicon revision is identified by a device revision code marked on the package. The code is of the format ##xx_#######, where "xx" denotes the silicon revision. If xx is “10”, then the silicon is revision 1.0 and the device part number (x) will have *no letter* (blank); if xx is “11”, then the silicon is revision 1.1 and for TMX the device part number (x) will have an "A", and for TMS the device number will have *no letter* (blank). if xx is “30”, then the silicon is revision 3.0 and the device part number (x) will have a "C", etc. Table 1 lists the information associated with each silicon revision.

Table 1. TMS320DM6467 Device Revision Codes

DEVICE REVISION CODE (xx) and (x)		SILICON REVISION	PART NUMBERS/COMMENTS
DIE PG CODE (xx)	DEVICE PART NUMBER (x)		
30	C	3.0	TM**X**320DM6467**C**ZUT, TM**S**320DM6467**C**ZUT
11	(blank)	1.1	TM**S**320DM6467ZUT
	A	1.1	TM**X**320DM6467**A**ZUT
10	(blank)	1.0	This silicon revision is available as TMX only. TMX320DM6467ZUT

2 Silicon Revision 3.0 Usage Notes and Known Design Exceptions to Functional Specifications

This section describes the usage notes and advisories that apply to silicon revision 3.0 of the TMS320DM6467 device.

Note: Two ***new*** features were added from Silicon Revision 1.1 to Silicon Revision 3.0: one in VPIF peripheral and the other in the UART peripheral.

- VPIF: Clipping Function for Output
- UART: CIR Demodulator Bypass Mode

For more detaied information on these new features, see the peripheral-specific User's Guide under the sections titled "*Clipping Function for Output*" for VPIF and "*CIR Demodulator Bypass Mode*" for UART.

2.1 Usage Notes for Silicon Revision 3.0

Usage notes highlight and describe particular situations where the device's behavior may not match presumed or documented behavior. This may include behaviors that affect device performance or functional correctness. These usage notes will be incorporated into future documentation updates for the device (such as the device-specific data sheet), and the behaviors they describe will not be altered in future silicon revisions.

2.1.1 PCI Cannot Burst More Than 32 Bytes When Used in Master Mode

On DM646x silicon revision 3.0 and earlier, the device PCI can operate as a PCI master and slave. As a slave, the device PCI responds to accesses initiated by an off-chip PCI master. As a master, the device PCI initiates transfers on the PCI bus. Usually, for memory read and write transfers, another DM646x master such as the EDMA is configured to move data to/from the PCI.

As a PCI master, the device PCI is capable of bursting only a maximum of up to 32 bytes. In other words, for memory transfers larger than 32 bytes, the device PCI initiates a transfer, transfers 32 bytes, stops the transfer, and then repeats. As a PCI slave, external PCI masters can burst an infinite amount of data to the DM646x PCI. Note that the PCI may insert wait states or generate a target retry if it cannot meet the latency requirement set forth by the PCI system. For example, a PCI access to DDR2 memory may stall due to other master accesses or because of a scheduled DDR2 memory refresh. In this case, the PCI generates a target retry until the DDR2 memory controller is ready to service the PCI request.

Because of this limitation, the DM646x PCI throughput will be lower in master mode than in slave mode. To avoid low throughput performance, external PCI masters should be used to move data to/from the DM646x PCI whenever possible.

2.1.2 VPIF Resynchronization After Disconnecting External Video Source

The VPIF module does not support resynchronization after disconnecting an external video source and the VP_ERRINT is not designed to flag a loss-of-sync event. In order to capture correctly, the VPIF has to be reset each time a source is reconnected.

One approach to reliably reset the VPIF is to have the DM646x device periodically poll the status register of the ADC which interfaces to the VPIF via the I2C bus. For example, on the DM646x EVM, the DM646x CPU can poll the TVP7002 or TVP5147 Sync Detect Status register via the I2C interface. When an invalid sync status is detected, the DM646x CPU disables the VPIF and continues polling periodically. When a valid sync status is detected, the VPIF is reset, reenabled, and reinitialized to capture.

Furthermore, if the ADC can send a toggle signal during a loss-of-sync, this signal can be connected directly to the GPIO of the DM646x to trigger a GPIO interrupt. Upon receiving the interrupt, the CPU can then reset the VPIF. This avoids the CPU from constantly reading the I2C bus.

2.2 Silicon Revision 3.0 Known Design Exceptions to Functional Specifications

Table 2. Silicon Revision 3.0 Advisory List

Title **Page**

Advisory 3.0.2 *Pin Muxing Functions: EMIFA-to-PCI Pin Mux Switching Resets Device*

Revision(s) Affected 3.0 and earlier

Details On the DM646x device, the EMIFA address 22 (EM_A[22]) and the $\overline{\text{PCI\_RST}}$ functions share the same pin (C10). The $\overline{\text{PCI\_RST}}$ function is internally connected to the chip reset. If the EM_A[22] function is low before the EMIFA-to-PCI pin mux switch happens (e.g., after any EMIFA boot mode execution), the device resets.

Workaround(s) Before switching the pin mux from EMIFA to PCI, drive EM_A[22] high by performing a dummy EMIFA read.

Advisory 3.0.3 — *DSP SDMA/IDMA: Unexpected Stalling and Potential Deadlock Condition When DSP L2 Memory Ports Used as RAM When L2 Memory Configured as Non-cache (RAM)*

Revision(s) Affected 3.0 and earlier

Details

Note: This advisory is ***not*** applicable if the DSP L2 memory is configured as 100% cache, ***or*** if L2 RAM and HDVICP0/1 RAM/Buffers (through the DSP SDMA port) *are not* accessed by IDMA or SDMA during run-time.

Note: Masters can access HDVICP0/1 RAM/Buffers either through HDVICP EDMA ports (that are connected to SCR1 through BR9 to BR14) or through HDVICP ports that are connected to DSP SDMA port via BR25, BR26, and SCR8 (as shown in the TMS320DM646x System Interconnect Block Diagram in *TMS320DM6467 SoC Architecture and Throughput Overview* Application Report (literature number SPRAAW4)).

The C64x+ Megamodule has a Master Direct Memory Access (MDMA) bus interface and a Slave Direct Memory Access (SDMA) bus interface. The MDMA interface provides DSP access to resources outside the C64x+ Megamodule (i.e., DDR2, EMIFA, HDVICP0/1 EDMA ports, PCI, ARM TCM, and VLYNQ remote memory). The MDMA interface is typically used for CPU/cache accesses to memory beyond the Level 2 (e.g., L2 RAM/Cache, HDVICP0/1 RAM/Buffers through DSP SDMA port) memory level. These accesses include cache line allocates, write-backs, and non-cacheable loads and stores to/from system memories. The SDMA interface allows other master peripherals, including ARM (data port), EDMA transfer controllers, HPI, USB2.0, ATA, EMAC, PCI, and VLYNQ, to access Level 1 Data (L1D), Level 1 Program (L1P), L2 DSP memories, and HDVICP0/1 RAM/Buffers (through DSP SDMA port). The DSP Internal DMA (IDMA) is a C64x+ Megamodule DMA engine used to move data between internal DSP memories (L1,L2) and/or the DSP peripheral configuration bus. The IDMA engine shares resources with the SDMA interface.

The C64x+ Megamodule has an L1D cache and L2 cache both implementing write-back data caches–it holds updated values for external memory as long as possible. It writes these updated values, called "victims", to external memory when it needs to make room for new data *or* when requested to do so by the application, or when a load is performed from a non-cacheable memory for which there is a set match in the cache (i.e., the non-cacheable line would replace a dirty line if cached). The L1D sends its victims to L2. The caching architecture has pipelining, meaning multiple requests could be pending between L1, L2, and MDMA. For more details on the C64x+ Megamodule and its MDMA and SDMA ports, see the *TMS320C64x+ Megamodule* Reference Guide (literature number SPRU871).

Ideally, the MDMA (dashed-dotted line in Figure 2) and SDMA/IDMA paths (dashed lines in Figure 2) operate independently with minimal interference. Normally MDMA accesses may stall for extended periods of time due to expected system level delays (e.g., bandwidth limitations, DDR2 memory refreshes). However, when using L2 as RAM, SDMA and IDMA accesses to L2/L1/HDVICP RAM/Buffers (through DSP SDMA port) may experience unexpected stalling in addition to the normal stalls seen by the MDMA interface. For latency-sensitive traffic, the SDMA stall can result in missing real-time deadlines. In a more severe case, the SDMA stall can produce a deadlock condition in the SoC. An IDMA stall cannot produce a deadlock condition.

Note: SDMA/IDMA accesses to L1P/D ***will not*** experience an unexpected stall if there are no SDMA/IDMA accesses to Level 2 memory ports (L2 RAM and HDVICP0/1 RAM/Buffers through DSP SDMA port). Unexpected SDMA/IDMA stalls to L1 happen *only* when they are pipelined behind Level 2 memory port accesses. Additionally, the deadlock scenario will be avoided if there are no SDMA accesses to the Level 2 memory ports.

Note: Figure 2 is provided for illustrative purposes and is incomplete for simplification. The IDMA/SDMA (dashed-lines) path could also go to L1D/L1P memories, and IDMA can go to DSP CFG peripherals. MDMA transactions can originate also from L1P or L1D through the L2 controller or directly from the DSP.

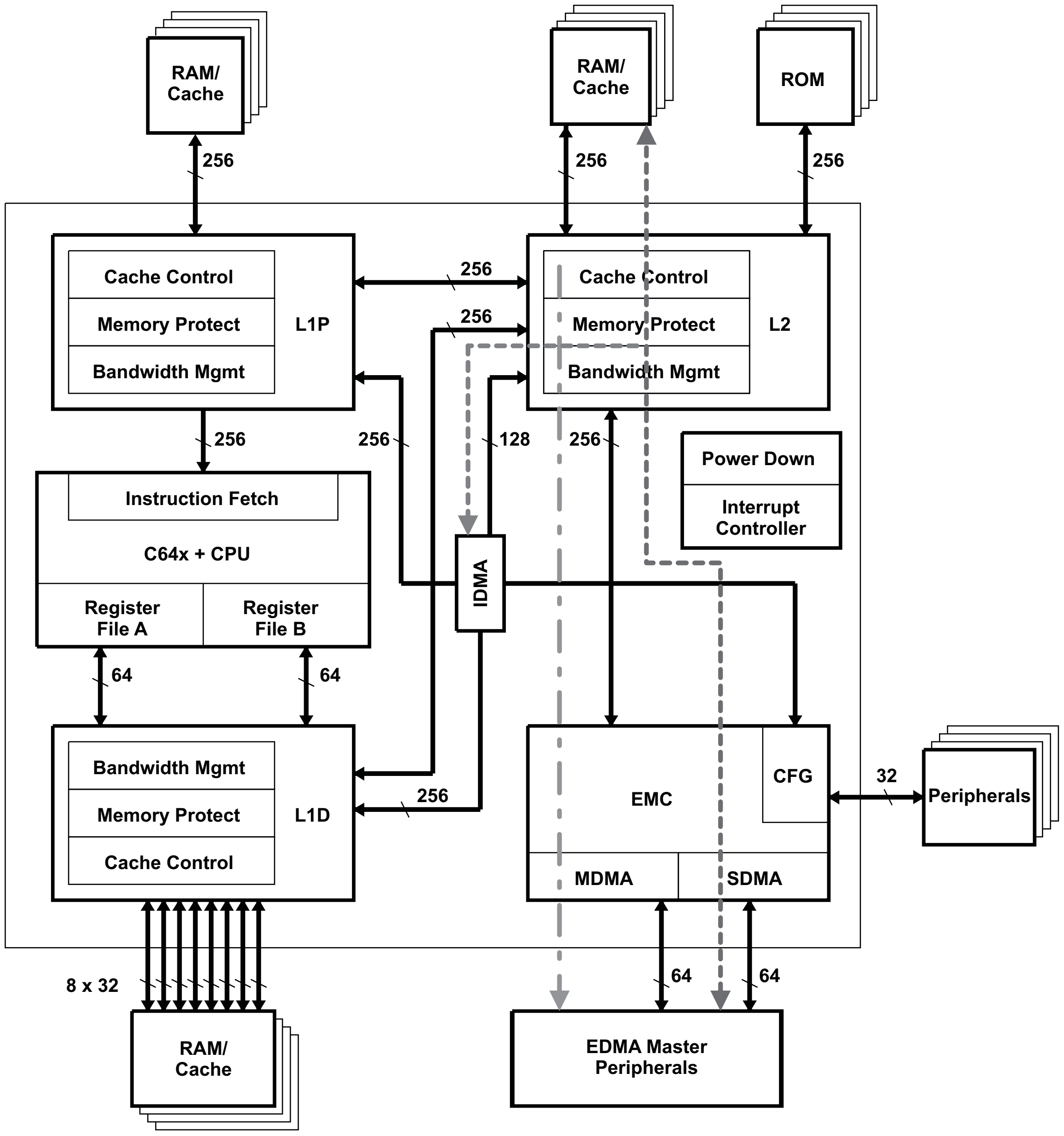


Figure 2. IDMA, SDMA, MDMA Paths

NOTES:

1. The dashed lines in Figure 2 represent SDMA/IDMA paths.
2. The dashed-dotted line in Figure 2 represents the MDMA path.

SDMA/IDMA stalls may occur during the following scenarios. Each of these scenarios describes expected normal DSP functionality, but the SDMA/IDMA access potentially exhibits additional unexpected stalling.

1. Bursts of writes to non-cacheable MDMA space (i.e., DDR2, EMIFA, ARM TCM, HDVICP0/1 EDMA ports, PCI, and VLYNQ remote). The DSP buffers up to 4 non-cacheable writes. When this buffer fills, SDMA/IDMA is blocked until the buffer is no longer full. Therefore, bursts of non-cacheable writes longer than three writes can stall SDMA/IDMA traffic.
2. Various combinations of L1 and L2 cache activity:
   (a) L1D read miss generating victim traffic to L2 (Cache or SRAM) or external memory. The SDMA/IDMA may be stalled while servicing the read miss and the victim. If the read miss also misses L2 cache, the SDMA/IDMA may be stalled until data is fetched from external memory to service the read miss. If the read access is to non-cacheable memory there will still potentially be an L1D victim generated even though the read data will not replace the line in the L1D cache.
   (b) L1D read request missing L2 (going external) while another L1D request is pending. The SDMA/IDMA may be stalled until the external memory access is complete.
   (c) L2 victim traffic to external memory during any pending L1D request. SDMA/IDMA may be stalled until external memory access and the pending L1D request is complete.

The duration of the IDMA/SDMA stalls depends on the quantity/characteristics of the L1/L2 cache and MDMA traffic in the system. In cases 2a., 2b., and 2c., stalling may or may not occur depending on the state of the cache request pipelines and the traffic target locations. These stalling mechanisms may also interact in various ways, causing longer stalls. Therefore, it is difficult to predict if and how long stalling will occur.

IDMA/SDMA stalling and any system impact is most likely in systems with excessive context switching, L1/L2 cache miss/victim traffic, and heavily loaded EMIFA.

Use the following procedure to determine if SDMA/IDMA stalling is the cause of real-time deadline misses for existing applications. Situations where real-time deadlines may be missed include loss of McASP samples and low peripheral throughput.

1. Determine if the transfer missing the real-time deadline is accessing L2, HDVICP0/1 RAM/Buffers through DSP SDMA port or L1D memory. If not, then SDMA/IDMA stalling is ***not*** the source of the real-time deadline miss.
2. Identify all SDMA transfers to/from Level 2 memory ports (L2 RAM or HDVICP0/1 RAM/buffers through DSP SDMA port) [e.g., EDMA transfer to/from L2 from/to an McASP, PCI, *or* HPI block transfer to/from L2, EDMA transfer to/from HDVICP0/1 RAM/Buffers]. If there are no SDMA transfers going into Level 2 memory ports, then SDMA/IDMA stalling is ***not*** the source of the problem.
3. Redirect all SDMA transfers to L2 memory to other memories using one of the following methods:
   (a) Temporarily transfer all the L2 SDMA transfers to L1D SRAM or ARM RAM.
   (b) If not all L2 SDMA transfers can be moved to L1D memory or ARM RAM memory, temporarily direct some of the transfers to DDR2 memory and keep the rest in L1D memory/ARM RAM memory. Still, there should be ***no*** L2 SDMA transfers.
   (c) If 'a.' and 'b.' are not possible, move the transfer with the real-time deadline to EMAC CPPI RAM. If the EMAC CPPI RAM is not big enough, a two-step mechanism can be used to page a small working buffer defined in EMAC CPPI RAM into a bigger buffer in L2 SRAM. The EDMA module can be setup to automate this double buffering scheme without CPU intervention for moving data from EMAC CPPI RAM. Some throughput degradation is expected when the buffers are moved to EMAC CPPI RAM.

   Note: EMAC CPPI RAM memory is only word-addressable. Therefore, EDMA transfers to/from EMAC CPPI RAM ***must*** have SRCBIDX/DSTBIDX = 4.

If real-time deadlines are still missed after implementing any of the options in Step 3, then IDMA/SDMA stalling is likely ***not*** the cause of the problem. If real-time deadline misses are solved using any of the options in Step 3, then IDMA/SDMA stalling ***is*** likely the source of the problem.

Note: The above Step 3 is applicable only to the SDMA accesses to L2 RAM.

Accesses to HDVICP RAM/Buffers via SDMA interface are typically driven by an application-specific requirement and cannot be redirected to some other memory. Additionally, for memory to memory transfers involving HDVICP, the issue would translate more into unexpected longer stalls/delays, and thereby performance degradation, not necessarily missing real-time deadlines. For accesses to HDVICP RAM/Buffers via the SDMA interface, if unexpected performance degradation is seen, use the guidelines under Method 2 in the Workaround(s) section.

Deadlock Scenario

As previously mentioned, a possible deadlock scenario is introduced in the presence of the SDMA stalls just described. This scenario occurs for certain masters connected to main data pipelined SCR1 either directly or indirectly through a bridge. For DM646x devices the masters that fall into this category, as shown in the TMS320DM6467 System Interconnect Block Diagram in the *TMS320DM6467 SoC Architecture and Throughput Overview* Application Report (literature number SPRAAW4), are:

- The ARM data port (ARM-D), connected to SCR1 through Bridge 2
- EDMA transfer controllers (EDMA TCs), connected to SCR1 directly
- PCI, connected to SCR1 through Bridge 3
- The masters HPI and ATA connected to a satellite SCR2, connected to SCR1 through Bridge 6
- The masters EMAC, USB, and VLYNQ connected to a satellite SCR3, connected to SCR1 through Bridge 7

If the following sequence of events occurs, then a deadlock situation might arise.

1. One of the following accesses occur:
   (a) ARM-D issues a write command to the DSP’s SDMA followed by a subsequent write command to slave memories DDR2, EMIFA, or HDVICP0/1 EDMA ports.
   (b) An EDMA TC issues a write command to the DSP’s SDMA followed by a subsequent write command to slave memories DDR2, EMIFA, HDVICP0/1 EDMA ports, or ARM TCM.
   (c) The PCI issues a write command to the DSP’s SDMA followed by a subsequent write command to slave memories DDR2 or ARM TCM.
   (d) Any master connected to Bridge 6 (HPI and ATA) issues a write command to the DSP’s SDMA and the same or different master connected to Bridge 6 issues a subsequent write command to slave memories DDR2 or ARM TCM.
   (e) Any master connected to Bridge 7 (EMAC, USB and VLYNQ) issues a write command to the DSP’s SDMA and the same or different master connected to Bridge 7 issues a subsequent write command to to slave memories DDR2, EMIFA, HDVICP0/1 R/W EDMA ports, or ARM TCM.

   The following is a list of slave memories corresponding to each of the Bridge 7 masters:

   - Master EMAC has access to slave memory DDR.
   - Master USB has access to slave memories DDR, AEMIF, and ARM TCM.
   - Master VLYNQ has access to slave memories DDR, AEMIF, HDVICP0/1 R/W EDMA ports, and ARM TCM.
2. The DSP’s SDMA asserts itself not ready and is unable to accept more write data, and a MDMA cache line writeback is initiated from DSP memory to DDR2 memory or to another slave memory (e.g., EMIFA).

In the above scenario it is possible for data phases from the write command issued to DDR2 (or EMIFA) to be stuck behind the data phases for the write to the DSP's SDMA in the SCR.

Therefore, if the DSP issues victim traffic to the same slave (DDR2 or EMIFA) then data associated with the victim traffic (#2) intended for DDR2 (or EMIFA) will be stuck behind write commands issued for #1. However, due to the MDMA/SDMA blocking issue, the SDMA traffic for #1 will be waiting for the MDMA traffic for #2 to finish, manifesting itself into a deadlock situation.

Note: If the slave memories (listed above in Step 1) are not cached in L2 (via the MAR bit settings) and there are no long distance read/writes to these memories from DSP then, the deadlock scenario will not arise (as there would be no MDMA accesses).

Workaround(s)

Method 1

For applications involving L2 RAM only, entirely eliminate IDMA/SDMA stalling stalling and potential for a deadlock condition using one or both of the following:

1. Configure the entire L2 RAM as 100% cache (e.g., move all data buffers to L1D/P, ARM RAM, EMAC CPPI memory, or external memory).

   Note: Some throughput degradation is expected when the buffers are moved to ARM RAM or EMAC CPPI RAM. Additionally, CPPI memory is only word-addressable. Therefore, EDMA transfers to/from EMAC CPPI RAM ***must*** have SRCBIDX/DSTBIDX = 4.
2. Eliminate all IDMA/SDMA access to L2 RAM during any time IDMA/SDMA stalling would have an impact (e.g., could preload data/code through IDMA/SDMA during system initialization/re-configuration).

Method 2

For applications involving IDMA/SDMA accesses to L2 RAM and/or HDVICP0/1 RAM/Buffers through DSP SDMA port, perform any of the following to reduce the IDMA/SDMA stalling system impact:

1. Improve system tolerance on DMA side (IDMA/SDMA/MDMA):
   - (a) Understand and minimize latency-critical SDMA/IDMA accesses to Level 2 memories or L1P/D.
   - (b) Directly reduce critical real-time deadlines, if possible, at peripheral and/or I/O level (e.g., increase word size and/or reduce bit rates on serial ports).
   - (c) Reduce DSP MDMA latency by:
     - (i) Increase priority of DSP access to DDR2/EMIFA such that MDMA latency of MDMA accesses causing stalls is minimized.
       Note: Other masters, such as VPIF, may have real-time deadlines that dictate higher priority than DSP.
     - (ii) Lower PR_OLD_COUNT bit field setting in the DDR2 memory controller's Burst Priority Register. Values ranging between 0x10 and 0x20 should give decent performance and minimize latency; lower values may cause excessive SDRAM row thrashing.
     - (iii) Do not perform EMIFA accesses using EMIFA WAIT handshaking during DSP run-time. Devices using WAIT potentially insert excessive latency to external memory accesses.
2. Minimize offending scenarios on DSP/Caching side:
   - (a) If the DSP performing non-cacheable writes is causing the issue, insert protected non-cacheable reads (as shown in the last list item f. below) every few writes to allow write buffer to drain.
   - (b) Avoid caching from slow memories such as, Asynchronous Memory. Instead, page the data via the EDMA from the off-chip Async Memory to L2 SRAM or SDRAM space before accessing the data from the DSP.

     Note: Paging cannot occur while real-time deadlines must be met.

(c) Use explicit cache commands to trigger cache write-backs during appropriate times (e.g., L1D Writeback All and L2 Writeback All).
    Do not use these commands when real-time deadlines must be met.

(d) Restructure program data and data flow to minimize the offending cache activity.
   - (i) Define the read-only data as “const.” The const C keyword tells the compiler that the array will not be written to. By default, such arrays are allocated to the “.const” section as opposed to BSS. With a suitable linker command file, the developer can link the .const section off-chip, while linking .bss on-chip. Because programs initialize .bss at run time, this reduces the program’s initialization time and total memory image.
   - (ii) Explicitly allocate lookup tables and writeable buffers to their own sections. The #pragma DATA_SECTION(label, “section”) directive tells the compiler to place a particular variable in the specified COFF section. The developer can explicitly control the layout of the program with this directive and an appropriate linker command file.
   - (iii) Avoid directly accessing data in slow memories (e.g., flash); copy at initialization time to faster memories.

(e) Modify troublesome code.
   - (i) Rewrite using DMAs to minimize data cache writebacks. If the code accesses a large quantity of data externally, consider using DMAs to bring in the data, using double buffering and related techniques. This will minimize cache write-back traffic and the likelihood of IDMA/SDMA stalling.
   - (ii) Re-block the loops. In some cases, restructuring loops can increase reuse in the cache and reduce the total traffic to external memory.
   - (iii) Throttle the loops. If restructuring the code is impractical, then it is reasonable to slow it down. This reduces the likelihood that consecutive SDMA/IDMA blocks “stack up” in the cache request pipelines resulting in a long stall.

(f) Protect non-cacheable reads from generating an SDMA stall by freezing the L1D cache during the non-cacheable read access(es). The following example code contains a function that protects non-cacheable reads, avoids blocking during the reads, and, therefore, avoids the deadlock state.

```
;; ======================================================================== ;;
;;  Long Distance Load Word                                                  ;;
;;                                                                           ;;
;;      int long_dist_load_word(volatile int *addr)                          ;;
;;                                                                           ;;
;;  This function reads a single word from a remote location with the L1D   ;;
;;  cache frozen.  This prevents L1D from sending victims in response to    ;;
;;  these reads, thus preventing the L1D victim lock from engaging for the  ;;
;;  corresponding L1D set.                                                   ;;
;;                                                                           ;;
;;  The code below does the following:                                       ;;
;;                                                                           ;;
;;      1.  Disable interrupts                                               ;;
;;      2.  Freeze L1D                                                       ;;
;;      3.  Load the requested word                                          ;;
;;      4.  Unfreeze L1D                                                     ;;
;;      5.  Restore interrupts                                               ;;
;;                                                                           ;;
;;  Interrupts are disabled while the cache is frozen to prevent affecting  ;;
;;  the performance of interrupt handlers.  Disabling interrupts during     ;;
;;  the long distance load does not greatly impact interrupt latency,       ;;
;;  because the CPU already cannot service interrupts when it's stalled by  ;;
;;  the cache.  This function adds a small amount of overhead (~20 cycles)  ;;
;;  to that operation.                                                       ;;
;;                                                                           ;;
;; ======================================================================== ;;

        .asg    0x01840044,     L1DCC           ; L1D Cache Control
        .global _long_dist_load_word
        .text
        .asmfunc
; int long_dist_load_word(volatile int *addr)
_long_dist_load_word:
        MVKL    L1DCC,          B4
        MVKH    L1DCC,          B4
||      DINT                                    ; Disable interrupts
||      MVK     1,              B5
        STW     B5,             *B4             ; \_ Freeze cache
        LDW     *B4,            B5              ; /
        NOP     4
        SHR     B5,     16,     B5              ; POPER -> OPER
||      LDW     *A4,            A4              ; read value remotely
        NOP     4
        STW     B5,             *B4             ; \_ Restore cache
        RET     B3
||      LDW     *B4,            B5              ; /
        NOP     4
        RINT                                    ; Restore interrupts
        .endasmfunc

;; ======================================================================== ;;
;;  End of file: ldld.asm                                                    ;;
;; ======================================================================== ;;
```

Perform the following to eliminate the potential for a deadlock condition:

1. Force the ARM-D to perform writes to either the DSP SDMA (L1, L2, and HDVICP RAM/Buffers through SDMA port) or slave memories (DDR2, EMIFA, or HDVICP0/1 EDMA ports), but not to both.
2. Force each EDMA TC to perform writes to either DSP SDMA (L1, L2, and HDVICP RAM/Buffers through SDMA port) or slave memories (DDR2, EMIFA, HDVICP0/1 EDMA ports, or ARM TCM), but not to both.
3. For PCI, HPI, ATA, EMAC, USB, and VLYNQ, do one of the following:
   (a) Force the completion of pending write commands to either DSP SDMA (L1, L2, and HDVICP RAM/Buffers through SDMA port) or slave memories before initiating writes to a different destination. Pending write commands from a particular master are forced to complete when the same master initiates a read from the same destination memory. **Note**: In the case of DDR2 and EMIFA, a read command only forces the completion of write commands within a 2KB-aligned window.
   (b) Force each master (PCI) or all masters in a group ([HPI and ATA] one group, [EMAC, USB, and VLYNQ] another group) to perform writes to either DSP SDMA memory space or slave memories, but not to both (For list of slave memories corresponding to each master, see the ***Deadlock Scenario*** section of this advisory).

   Note: In the case of group of masters, for example [HPI and ATA] as a group (connected to Bridge 6), both of these masters must only perform writes to either DSP SDMA or slave memories, but not to both. For example, if HPI writes to DSP SDMA memory and ATA writes to DDR2 memory, the potential for the deadlock condition is still present. The same condition applies for other group of masters [EMAC, USB, and VLYNQ] connected to Bridge 7.

Note: If the slave memories (listed in the steps above) are not cached in L2 (via the MAR bit settings) and there are no long-distance read/writes to these memories from DSP, then the deadlock scenario will not arise (as there would be no MDMA accesses) and no need to perform the workaround.

Advisory 3.0.4 *Glitch on I2C Bus During Device Power-Up Sequence*

Revision(s) Affected: 3.0 and earlier

Details: The suggessted power-up sequence was 3.3 V→1.8 V→1.2 V. However, there is a small glitch on I2C bus during this power-up sequence. If the board design is already done with this power-up sequence (3.3 V→1.8 V→1.2 V), there is no need to change. This glitch should not be an issue because it is not a valid I2C bus transition (i.e., the glitch does not match either the START or STOP condition).

Workaround: Reverse the power-up sequence, that is, 1.2 V→1.8 V→3.3 V.

Advisory 3.0.16 *HDVICP – H.263 Encode: Quantized AC Coefficient Clipping*

Revision(s) Affected: 3.0 and earlier

Details: This low impact issue is rare and occurs only when ***all*** the following conditions are met:

- Encode
- H.263

In H.263 or MPEG4 (short_video_header = 1) encoding, the quantized AC coefficient should be clipped within a range of [-127:127]. But, the HD-VICP CALC QiQ function only supports the range of [- (2^N-1):2^N-1] (N = 8 to 15). Therefore, the minimum clipping range will be [-255:255].

This saturation can be mitigated by clipping the QP to a minimum value of 8 without perceivable quality loss.

Workaround: Control the QPmin for the encoder.

The DCT transform coefficient value will be in the range of -1024 to 1023, so the minimum value QP can be 8 to limit the quantized AC coefficient within the range of [-127:127].

Advisory 3.0.19 *DMA Access to L2 SRAM May Stall When the C64x+ CPU Command Priority is Lower Than or Equal to the DMA Command Priority*

Revision(s) Affected: 3.0 and earlier

Details: **Note:** DMA refers to all non-CPU requests. This includes Internal Direct Memory Access (IDMA) requests and all other system DMA master requests via the Slave Direct Memory Access (SDMA) port.

The C64x+ Megamodule uses a bandwidth management (BWM) system to arbitrate between the DMA and CPU requests issued to L2 RAM. For more information on the BWM feature, see the *TMS320C64x+ DSP Megamodule* Reference Guide (Literature Number: SPRU871). The BWM arbitration grants L2 bandwidth based on programmable priorities and contention-cycle-counters. The contention-cycle-counters count the number of cycles for which the associated L2 requests are blocked by higher-priority requests. When the contention-cycle-counter reaches a programmed threshold (MAXWAIT), the associated L2 request is granted a slice of L2 bandwidth. This prevents indefinite blocking of lower-priority requests when faced with the continuous presence of higher-priority requests.

Ideally, the BWM arbitration will grant equal L2 bandwidth between equal priority DMA and CPU requests. Instead, when requests arrive at the BWM such that the CPU priority is *lower than or equal to* the DMA priority, the bandwidth is always granted in favor of the CPU over the DMA. In the case of successive CPU requests, it is possible for the CPU to block all DMA requests until the CPU traffic subsides. Figure 3 shows a high-level diagram of the arbitration scheme used for L2 RAM requests.

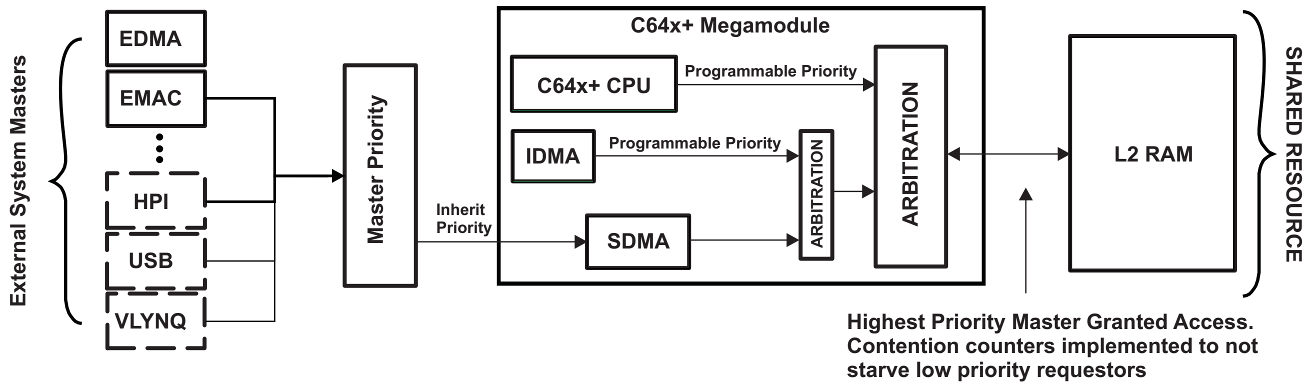


Figure 3. Priority Arbitration Scheme for L2 RAM

When the SDMA has finished sending all of its commands to the L2 controller, the C64x+ Megamodule drops the effective transfer priority to seven. The L2 Controller uses this effective priority to arbitrate the SDMA command with the CPU.

This happens regardles of what the actual SDMA priority is. This means that if the CPU Priority is equal to seven, then it can also trigger the issue highlighted by this advisory.

Workaround: Configure the DMA and CPU requests to different priority levels such that the CPU priority level is higher than the DMA priority level. Priority seven should not be used for the CPU. There is no penalty for setting the IDMA and SDMA priorities equal to each other.

The following table highlights which priority combinations are affected and what combinations are valid:

Table 3. Allowable CPU and SDMA Priorities

CPU PRIORITY	SDMA PRIORITIES ALLOWED
0	Not allowed; Affected by Advisory Issue
1	0
2	0-1
3	0-2
4	0-3
5	0-4
6	0-5
7	Not allowed; Affected by Advisory Issue

Pseudo code provided below highlights how the various requestor priorities can be configured:

CPU request priority is programmed within the CPUARBU register:

```
/** Pseudo code only **/

        Uint32 *CPUARBU;

        CPUARBU = ( Uint32 * ) ( 0x01841000 );

        /* Set priority different from IDMA/SDMA */
        *CPUARBU = [CPU_PRIORITY];
```

IDMA request priority is programmed within the IDMA1_COUNT register

```
/** Pseudo code only **/

        Uint32 *IDMA1_SRC, *IDMA1_DST;
        Uint32 *IDMA1_CNT;

        IDMA1_SRC = ( Uint32 * ) ( 0x01820108 );
        IDMA1_DST = ( Uint32 * ) ( 0x0182010C );
        IDMA1_CNT = ( Uint32 * ) ( 0x01820110 );

        *IDMA1_SRC = sourceAddress;
        *IDMA1_DST = destinationAddress;

        /* Set IDMA priority different from CPU */
        *IDMA_CNT = ( [IDMA_PRI] << [IDMA_PRI_SHIFT] ) | buffsize ;
```

SDMA request priority is inherited from the MSTPRIn registers

```
/** Pseudo code only **/

        Uint32 *MSTPRI1, *MSTPRI2;

        MSTPRI1 = ( Uint32 * ) ( 0x01C40040 );
        MSTPRI2 = ( Uint32 * ) ( 0x01C40044 );

        /* Set SDMA master priorities different from CPU */
        *MSTPRI1 = [MAST_PRI] << [MAST_SHIFT];
        *MSTPRI2 = [MAST_PRI] << [MAST_SHIFT];
```

Advisory 3.0.20 *HDVICP—H.264 Encode/Decode: Intra 16 x 16 Plane MB Values ≥ 512 are Incorrect*

Revision(s) Affected: 3.0 and earlier

Details: During the calculation of the intra prediction block, pixel values can reach 512 or higher. However, the computation engine has precision only enough to hold values equal to 511 or less.

This issue can occur only when ***all*** the following conditions are met:

- H.264 Encode or Decode
- Intra_16x16_Plane mode MB
- Luma blocks *only*; it does not affect Chroma blocks
- If the prediction block generated using the Intra_16x16_Plane mode has a pixel value of 512 or higher

The Artifacts by this advisory are:

- In Encode, an incorrect (but legal) bitstream will be generated, and a mismatch will happen between the Encoder and Decoder resulting in signficant quality degradation.
- In Decode, the reconstructed macroblock will be incorrect.
- If the following MBs are intra referring to this MB, the error is propagated spatially within the picture.
- If the following pictures refer to this MB, the error is propagated temporally to the following pictures.

This advisory is content dependent. Typically, generating Intra Prediction Values ≥ 512 is rare but can occur; therefore, the suggested workarounds below should be used. A potential exception could be for a closed-system decoder where the user can ensure that the encoder does not use Intra_16x16_Plane mode.

Workaround(s):

Encode

Turn off the Intra_16x16_plane mode evaluation by HDVICP IPE. Typically, the quality degradation for removing this mode is relatively small.

Decode

During a MB pipeline slot, a CALC is done, a workaround by CPU, and CALC again with corrected intra prediction macroblock (see Figure 4).

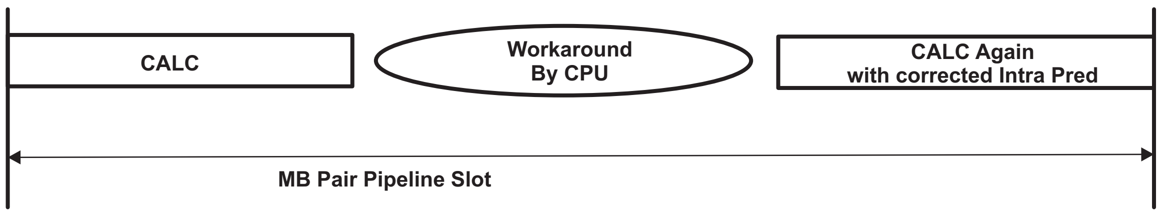


Figure 4. MB Pair Pipeline Slot Example

If the MB is of intra 16x16 plane mode, perform the following steps:

1. Let CALC run.
2. Wait for CALC to complete
3. After CALC completes, switch BFSW for iprdbuf : CALC → DMA
4. Check and correct Intra Prediction (for more details, see the *Check and Correct Intra Prediction* section below)
5. Switch BFSW for iprdbuf : DMA → CALC
6. Switch BFSW for calcmbuf : CALC → DMA
7. Set up CALC commands with intra mode for IQ/IT and inter mode for prediction (for more details, see the *Set up CALC Commands with Intra mode for IQ/IT and Inter Mode for Prediction* section below
8. If MBAFF, retrigger DMA for upper row data
9. Switch BFSW for calcmbuf : DMA → CALC
10. Switch BFSW for reconbuf : LPF → DMA
11. Read left reconstructed column
12. Switch BFSW for reconbuf : DMA → LPF
13. Switch BFSW for calcsbuf : CALC → DMA
14. Set back left reconstructed column
15. Switch BFSW for calcsbuf : DMA → CALC
16. Re-run CALC
17. Wait for CALC to complete

Check and Correct Intra Prediction

Read Pred[0,0], Pred[15,0], Pred[0,15], Pred[15,15], Pred[7,7] and Pred[7,8] (or Pred[8,7]) from iprdbuf

/* This workaround uses the fact that 1D slope is constant on a Plane. */

```
/* This workaround uses the fact that 1D slope is constant on a Plane. */

// right top pixel is wrong
if (Pred[15,0] == 0x00 && Pred[0,15] < Pred[7,8] (or Pred[8,7])) {
    Pred[15,0] = 0xFF;
}
// left bottom pixel is wrong
else if (Pred[0,15] == 0x00 && Pred[15,0] < Pred[7,8] (or Pred[8,7])) {
    Pred[0,15] = 0xFF;
}
// right bottom pixels are wrong
else if (Pred[15,15] == 0x00 && Pred[0,0] < Pred[7,7]) {
    Pred[15,15] = 0xFF
    if (Pred[11,15] == 0x00) Pred[11,15] = 0xFF;
    if (Pred[12,14] == 0x00) Pred[12,14] = 0xFF;
    if (Pred[12,15] == 0x00) Pred[12,15] = 0xFF;
    if (Pred[13,13] == 0x00) Pred[13,13] = 0xFF;
    if (Pred[13,14] == 0x00) Pred[13,14] = 0xFF;
    if (Pred[13,15] == 0x00) Pred[13,15] = 0xFF;
    if (Pred[14,12] == 0x00) Pred[14,12] = 0xFF;
    if (Pred[14,13] == 0x00) Pred[14,13] = 0xFF;
    if (Pred[14,14] == 0x00) Pred[14,14] = 0xFF;
    if (Pred[14,15] == 0x00) Pred[14,15] = 0xFF;
    if (Pred[15,11] == 0x00) Pred[15,11] = 0xFF;
    if (Pred[15,12] == 0x00) Pred[15,12] = 0xFF;
    if (Pred[15,13] == 0x00) Pred[15,13] = 0xFF;
    if (Pred[15,14] == 0x00) Pred[15,14] = 0xFF;
}
```

Set up CALC Commands With Intra Mode for IQ/IT and Inter mode for Prediction

To do the 2nd run, set CALC commands per Table 4 and Table 5. MB mode and Q/IQ information will need to be modified.

Table 4. Format of MB-mode for Spatial Intra Prediction

MB-MODE	
BIT NO.	**DESCRIPTION**
31	TQ Bypass-mode **[0]** :Disable, [1] :Enable.
30:28	Reserved.
27	Chroma DC-Transform (2x2) [0] :Disable, **[1]** : Enable.
26	Luma DC-Transform (4x4) [0] :Disable, **[1]** :Enable.
25:24	Scaling-timing [00] : Do not sue Scaling -function. [01] : Scaling per Luma/Chroma. **[10]** : Scaling per Color-Block. [01] : Scaling per Block.
23:20	Reserved.
19:18	Transform size of Block [5] [00] :8x8 , [01] :8x4 , [10] :4x8 , **[11]** :4x4
17:16	Transform size of Block [4] [00] :8x8 , [01] :8x4 , [10] :4x8 , **[11]** :4x4
15:14	Transform size of Block [3] [00] :8x8 , [01] :8x4 , [10] :4x8 , **[11]** :4x4
13:12	Transform size of Block [2] [00] :8x8 , [01] :8x4 , [10] :4x8 , **[11]** :4x4
11:10	Transform size of Block [11] [00] :8x8 , [01] :8x4 , [10] :4x8 , **[11]** :4x4
9:8	Transform size of Block [0] [00] :8x8 , [01] :8x4 , [10] :4x8 , **[11]** :4x4
7	Reserved.
6:4	Chroma Block-mode [000] :Reserved , **[001]** :Intra 8x8 , [010] :Reserved , [011] :Intra w/o Pred. , [100] :Inter , [101] : Reserved , [110] :Reserved , [111] :Skip.
3	Reserved.
2:0	Luma Block-mode [000] :Intra 16x16 , [001] :Intra 8x8 , [010] :Intra 4x4 , [011] :Intra w/o Pred. , **[100]** :Inter , [101] : Reserved , [110] :Reserved , [111] :Skip.

For Table 5, set Inter shift-scale value = Intra shift-scale value.

Table 5. Q/IQ Information

PARAMETER FOR Q/IQ		
ADDRESS OFFSET	**BIT NO.**	**DESCRIPTION**
0x08	63:24	Reserved.
	23:16	**Shift-scale for IQ Intra-DC-Y**
	15:8	**Shift-scale for IQ Inter-DC-Y**
	7:0	Shift-scale for IQ AC-Y
0x10	63:56	Reserved.
	55:48	**Shift-scale for IQ Intra-DC-Cr**
	47:40	**Shift-scale for IQ Inter-DC-Cr**
	39:32	Shift-scale for IQ AC-Cr
	31:24	Reserved.
	23:16	**Shift-scale for IQ Intra-DC-Cb**
	15:8	**Shift-scale for IQ Inter-DC-Cb**
	7:0	Shift-scale for IQ AC-Cb
0x18	63:24	Reserved.
	23:16	**Shift-scale for Q Intra-DC-Y**
	15:8	**Shift-scale for Q Inter-DC-Y**
	7:0	Shift-scale for Q AC-Y
0x20	63:56	Reserved.
	55:48	**Shift-scale for Q Intra-DC-Cr**
	47:40	**Shift-scale for Q Inter-DC-Cr**
	39:32	Shift-scale for Q AC-Cr
	31:24	Reserved.
	23:16	**Shift-scale for Q Intra-DC-Cb**
	15:8	**Shift-scale for Q Inter-DC-Cb**
	7:0	Shift-scale for Q AC-Cb
0x28	63:32	Round coefficient [1]
	31:0	Round coefficient [0]

This advisory has been fixed in TI Decoder S/W Version 01.10.01 or later.

3 Silicon Revision 1.1 Usage Notes and Known Design Exceptions to Functional Specifications

This section describes the usage notes and advisories that apply to silicon revision 1.1 of the TMS320DM6467 device.

3.1 Usage Notes for Silicon Revision 1.1

Usage notes highlight and describe particular situations where the device's behavior may not match presumed or documented behavior. This may include behaviors that affect device performance or functional correctness. These usage notes will be incorporated into future documentation updates for the device (such as the device-specific data sheet), and the behaviors they describe will not be altered in future silicon revisions.

Silicon revision 1.1 applicable usage notes have been found on a later silicon revision. For more details, see Section 2.1, *Usage Notes for Silicon Revision 3.0*.

3.2 Silicon Revision 1.1 Known Design Exceptions to Functional Specifications

Some silicon revision 1.1 applicable advisories have been found on a later silicon revision. For more details, see Section 2.2, *Silicon Revision 3.0 Known Design Exceptions to Functional Specifications*.

Table 6. Silicon Revision 1.1 Advisory List

Title	Page
Advisory 1.1.1 —HPI: HPI Signals are Driven When Device is in Reset	27
Advisory 1.1.5 —ARM ROM UART Boot Loader (RBL) Uses Only 16KB of Internal RAM	28
Advisory 1.1.6 —Write Protect Should be Enabled for NAND Flash Boot	29
Advisory 1.1.7 —EM_WAIT[5:3] Signals Should Not Toggle During NAND Boot	29
Advisory 1.1.8 —HDVICP – VC1 Decode: B Frame Intensity Compensation is not Implemented Correctly	30
Advisory 1.1.9 —HDVICP – VC1 Decode/Encode: AC Prediction is Incorrect Due to Improper AC Scaling	31
Advisory 1.1.10 —HDVICP – VC1 Decode: Quantization Information Decoded by ECD is Incorrect	33
Advisory 1.1.11 —HDVICP – VC1 Decode – Zigzag Table Selection: ECD Selects Incorrect DC Coefficient	34
Advisory 1.1.12 —HDVICP – VC1 Decode – MV Calculations: ECD Does Not Decode Proper Motion Vectors	35
Advisory 1.1.13 —HDVICP – VC1 Decode – MV calculations: ECD Decodes Motion Vectors Incorrectly	36
Advisory 1.1.14 —HDVICP – VC1 Encode: ECD Encodes Incorrect Bitstream When CBP = 0	37
Advisory 1.1.15 —HDVICP – VC1 Encode: ECD Encodes Incorrect Bitstream When Block is Less Than 8 x 8	38
Advisory 1.1.17 —HDVICP – H.264 Decode: Constrained Intra-Prediction –Affects High Profile Streams	39
Advisory 1.1.18 —HDVICP – H.264 Decode: 128 Weight Value Weighted Prediction Not Supported	41

Advisory 1.1.1 *HPI: HPI Signals are Driven When Device is in Reset*

Revision(s) Affected 1.1 and earlier

Details All HPI signals are ***not*** 3-stated when the device is held in reset ($\overline{\text{RESET}}$ and $\overline{\text{POR}}$ held low). The following HPI signals are driven when the device is held in reset ($\overline{\text{RESET}}$ and $\overline{\text{POR}}$ held low). This can result in bus contention with HPI host when the device is in reset.

Workaround(s) Isolate these HPI signals from the host by using a FET bus isolation switch (CBT).

SIGNAL NO.	SIGNAL NAME	SIGNAL STATE AT RESET ($\overline{\text{RESET}}$ and $\overline{\text{POR}}$ HELD LOW)
B3	$\overline{\text{PCI\_DEVSEL}}$/HCNTL1/EM_BA[1]	Held High
D5	$\overline{\text{PCI\_STOP}}$/HCNTL0/$\overline{\text{EM\_WE}}$	Held High
A5	$\overline{\text{PCI\_CBE3}}$/HR/$\overline{\text{W}}$/$\overline{\text{EM\_CE3}}$	Held High
E6	$\overline{\text{PCI\_TRDY}}$/HHWIL/EM_A[16]/(ALE)	Held Low
C4	$\overline{\text{PCI\_CBE2}}$/$\overline{\text{HDS2}}$/$\overline{\text{EM\_CS2}}$	Held High
B2	$\overline{\text{PCI\_SERR}}$/$\overline{\text{HDS1}}$/$\overline{\text{EM\_OE}}$	Held High

This advisory will be fixed in a future *major* silicon revision.

Advisory 1.1.5 *ARM ROM UART Boot Loader (RBL) Uses Only 16KB of Internal RAM*

Revision(s) Affected: 1.1 and earlier

Details: The ARM ROM Boot Loader (RBL) is using only 16KB of internal RAM instead of 32KB for UART boot. With this limitation, the UART boot *does not* work if the UBL size is greater than 14KB (2KB out of 16KB is reserved for RBL). Figure 5, shows what a memory map with this limitation looks like:

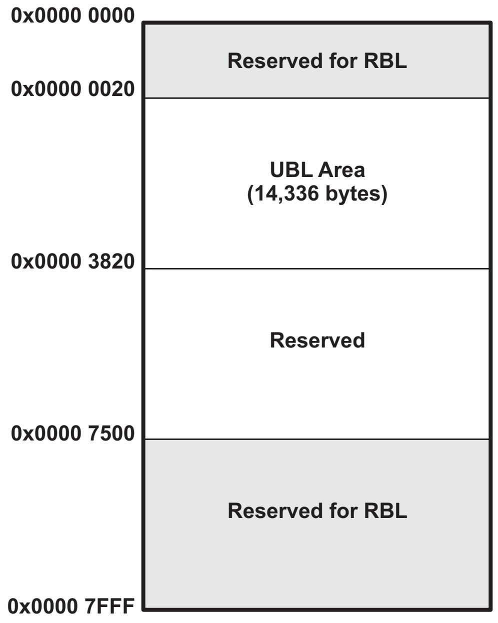


Figure 5. ROM Boot Loader and User Boot Loader Memory Map

The starting address of the UBL (User Boot Loader) is at 0x0020 and can be loaded between 0x0020 and 0x381F.

Workaround: If the UBL size is less than 14KB, the UART Boot works fine. If the UBL is size is greater than 14KB, the user needs to develop a secondary boot loader (SBL) of a size less than 14KB. Load the SBL first using the UART boot to RAM between the address range of 0x0020 and 0x381F. The SBL in turn should take care of loading the actual UBL to RAM or DDR2 (Note: The SBL should configure the DDR2 before loading to the DDR2).

This advisory will be fixed in a future *major* silicon revision.

Advisory 1.1.6 *Write Protect Should be Enabled for NAND Flash Boot*

Revision(s) Affected: 1.1 and earlier

Details: Write Protect should be enabled during NAND Boot; otherwise, the ARM RBL (ROM Boot Loader) *incorrectly* detects the NAND ready status.

Workaround: Enable the NAND Flash Write Protect during boot. This requires the user to drive the write protect pin on the NAND Flash. If the NAND Flash is used for more than a boot device, then the user needs to design a workaround to control the write protect pin on the NAND Flash using a GPIO pin.

This advisory will be fixed in a future *major* silicon revision.

Advisory 1.1.7 *EM_WAIT[5:3] Signals Should Not Toggle During NAND Boot*

Revision(s) Affected: 1.1 and earlier

Details: The EM_WAIT[5:3] signals should not toggle during NAND boot. They should stay either low *or* high during the NAND boot; otherwise, the ARM RBL (ROM Boot Loader) *incorrectly* detects the the NAND ready status for a READ.

Workaround: Keep the EM_WAIT[5:3] signals either low *or* high during the NAND boot. Make sure there is no cross coupling/noise from the PCB that could cause a rising edge on EM_WAIT[5:3] signals.

This advisory will be fixed in a future *major* silicon revision.

Advisory 1.1.8 — *HDVICP – VC1 Decode: B Frame Intensity Compensation is not Implemented Correctly*

Revision(s) Affected: 1.1 and earlier

Details: This low impact issue is rare and occurs only when **all** the following conditions are met:

- Universal Decode
- Main Profile
- Progressive/Interlaced
- B picture

The VC1 Main/Advanced profile streams may contain Intensity compensation. According to the VC-1 standard (*Proposed SMPTE Standard for Television: VC-1 Compressed Video Bitstream Format and Decoding Process*, SMPTE 421M, 2005-06-30), there are cases where intensity compensation should be used in decoding the B picture (see Section 8.3.8, *Intensity Compensation* of the VC-1 Standard Document). When the intensity compensation is ON for the backward reference frame, all the pixels in the forward reference frame are intensity compensated before using them for interpolation. So at any instant for the B frame, only one reference frame (forward) is intensity compensated. In the DM6467 device, intensity compensation is done at the MB loop instead of done at the frame level and is handled by the MC engine. In the case of B frames, the MC engine applies intensity compensation for both the forward and backward frames instead of doing intensity compensation for only the forward reference frame.

The intensity compensation tool is used when background intensity change happens. Since it is a complex tool, the encoder does not use it. Hence, the streams with this behavior are not generally available.

Workaround: Intensity compensation is controlled dynamically at the Macroblock (MB) level.

Macroblocks in B frames can be used in the following modes:

- Forward prediction only
- Backward prediction only
- Direct or Interpolated prediction

Intensity compensation is forcibly switched OFF in the case where backward prediction, interpolated prediction, and all the pixels in the forward reference frame are intensity compensated by using the DSP before giving the information to the MC engine as input.

This advisory will be fixed in a future *major* silicon revision.

Advisory 1.1.9 *HDVICP – VC1 Decode/Encode: AC Prediction is Incorrect Due to Improper AC Scaling*

Revision(s) Affected: 1.1 and earlier

Details: This low impact issue is rare and occurs only when ***all*** the following conditions are met:

- Universal Decode
- Simple/Main/Advanced Profile
- Progressive/Interlaced

In the VC-1 Standard before doing the AC prediction, AC scaling is applied to AC predictor in order to align the current quantization parameter. AC scaling is done by using the following formula:

$AC_P \ncong [AC_P \times STEP_P \times DQScale [STEP_C] + 0x20000] >> 18$

However, the HDVICP CALC does not add the rounded-off value (0x20000) like in the above formula; so, use the following formula for the HDVICP CALC value:

$AC_P \ncong [AC_P \times STEP_P \times DQScale [STEP_C]] // 2\text{^}18$

Because of that, the CALC AC prediction are incorrect when:

- AC prediction is ON
- Q parameter is different between the predictor position and the current position
- Predictor AC_P is negative value
- AC_P x $STEP_P$ x DQScale [$STEP_C$] = …X_XX10_0000_0000_0000_0000

The S/W workaround for such cases will have no performance impact. Moreover, the problem is self correcting on the next I frame.

Workaround(s): There are two **Software Workarounds**:

Note: The Software Method 1 process should be done only for boundary blocks in the Macro block.

Software Method 1: Modify AC predictors in CALC sbuffer before CALC trigger.

Before the CALC trigger perform the following steps:

1. Switch Residual buffer to DSP.
2. Check whether the AC prediction is ON and that the Q parameter is different between the predictor position and the current position.
3. If steps 1 and 2 are met, then Switch CALC sbuffer to DSP and check all the predictors to see whether they are negative.
4. Check for the condition AC_P x $STEP_P$ x DQScale [$STEP_C$] = …X_XX10_0000_0000_0000_0000
5. If conditions 2 through 4 are met, then add 1 to all the negative predictors
6. Switch back the CALC Sbuffer and the Residual buffer to CALC

Software Method 2: Do AC prediction in software by using the DSP.

Before the CALC triggers perfomr the following steps:

1. Switch the Residual buffer to DSP and check that AC prediction is ON for each block.
2. When the AC prediction is ON, Switch CALC Sbuffer to DSP, and do AC prediction by using the appropriate predictors in the CALC Sbuffer.
3. Modify the AC coefficients in the Residual buffer with the new values.
4. Switch off the AC prediction flag in the Residual buffer for that block.
5. Switch back the Residual buffer to ECD
6. Switch back the CALC Sbuffer to CALC

This advisory will be fixed in a future *major* silicon revision.

Advisory 1.1.10 *HDVICP – VC1 Decode: Quantization Information Decoded by ECD is Incorrect*

Revision(s) Affected: 1.1 and earlier

Details: This low impact issue is rare and occurs only when ***all*** the following conditions are met:

- Universal Decode
- Simple/Main/Advanced Profile
- Progressive
- P/B picture
- DQUANTFRM = 1

In general when CBPCY is equal to zero, there is no need for the transform and quantization information for that particular Macro block. However, the VC-1 standard says that the encoders may encode the transform and quantization information when the more_present_flag is 1 and CBPCY is zero. When CBPCY is zero, the ECD engine treats the more_present_flag as 0 and doesn't decode the MQDIFF or the ABSMQ information. The ECD engine decodes the TTMB information but due to the incorrect bitstream, the TTMB information may be incorrect.

Generally, the encoder does not encode syntax elements if the CBPY is zero. Moreover, the problem is self correcting on the next I frame.

Workaround: For this advisory, it has been observed that the ECD takes only ~250 cycles instead of ~800 cycles. Therefore, immediately after the ECD task switch to the ECD auxiliary buffer, check for the following conditions:

1. Bit stream is progressive P/B
2. Frame level Quant parameter "DQUANTFRM " is equal to 1
3. Macro block is Inter
4. Macro block is not skipped
5. CBPCY is equal to 0

When conditions 1 though 5 are ***all*** met, the DSP has to do the MB header parsing and set the correct bit stream pointers.

Note: To implement the workaround, the ECD should be operated in 1 MB mode and the DSP should take care of the stream buffer fullness.

This advisory will be fixed in a future *major* silicon revision.

Advisory 1.1.11 ***HDVICP – VC1 Decode – Zigzag Table Selection: ECD Selects Incorrect DC Coefficient***

Revision(s) Affected: 1.1 and earlier

Details: This low impact issue is rare and occurs only when ***all*** the following conditions are met:

- Universal Decode
- Simple/Main Profile
- Progressive/Interlaced
- I picture

The default DC value is assumed when the surrounding block DC coef is not available for calculating the DC coef, prediction direction, and zigzag scan. The ECD incorrectly calculates the DCdefault value for Simple/Main profile Intra frames when overlap filtering is ON for the MB. When there is no Left DC coef available (i.e., 1st MB Y0, Y2 in a row), the ECD calculates the DC default normally instead of biasing to zero. Therefore, when the upper DC coef is equal to zero or incorrectly calculated as the DCdefault value, the ECD doesn’t decode the proper DC coefficients.

This error occurs only in I frame for MB falling on the left picture edge. Since this condition is very infrequent in a GOV, the impact is low. Also, the DSP load for an I frame is a lot less compared to the P frame.

Workaround: **Software Workaround:**

Immediately after the ECD task, check to make sure the following conditions met:

- Stream is Simple/Main profile
- Picture type is Intra
- Overlap filter is ON
- 1st MB of a row

When all the above conditions are met, switch the ECD residual buffer to DSP and to the left Boundary blocks (Y0, Y2, Cb, Cr). Once complete, check whether the top DC coef is 0 or the incorrect DCdefault value.

- When the top DC coefficient is zero, change the residual coefficient scan order from horizontal to vertical.
- When the top DC coefficient is equal to the incorrect DCdefault value, recalculate the proper prediction direction and the DC coefficient.

NOTE: The DSP will have to check the other dependent blocks (Y1, Y3…) when the top DC coefficient is equal to the incorrect DCDefault value since DC coefficients of the corresponding blocks are predicted from the Y0, Y2, Cb, Cr blocks.

This advisory will be fixed in a future *major* silicon revision.

Advisory 1.1.12 *HDVICP – VC1 Decode – MV Calculations: ECD Does Not Decode Proper Motion Vectors*

Revision(s) Affected: 1.1 and earlier

Details: This low impact issue is rare and occurs only when ***all*** the following conditions are met:

- Universal Decode
- Advanced Profile
- Interlaced
- B picture
- MV RANGE of Anchor frame is bigger than current B picture
- MB type is forward or backward not direct or bi-Pred

The motion vectors are calculated by using the differential motion vector and the predicted motion vector as shown in the following formula:

mv = (dmv + pmv) smod(range) [see the VC-1 Standard Section 10.3.5.4.4.1, *Luma Motion Vector Reconstruction*]

The ECD incorrectly does the “smod” operation when the MVRANGE of the current frame and the predicted MVRANGE are different.

For example in the B frame Direct MB case, MVs are calculated by using the co-located MB motion vector data.

This error occurs only for the B frames when the MVRANGE of previous frame is different from present frame. Generally, the practical encoder does not change the MVRANGE across the frame level.

Workaround: In the case of P and B frames immediately after the ECD task, check the Motion vectors of neighboring MB for the condition "ABS(mv) > MV RANGE".

When this condition is met, do the Motion vector calculations with "DSP" and update the ECD MB data in both the ECD auxiliary and ECD work buffers.

This advisory will be fixed in a future *major* silicon revision.

Advisory 1.1.13 *HDVICP – VC1 Decode – MV calculations: ECD Decodes Motion Vectors Incorrectly*

Revision(s) Affected: 1.1 and earlier

Details: This low impact issue is rare and occurs only when ***all*** the following conditions are met:

- Universal Decode
- Simple/Main/Advanced Profile
- Progressive/Interlaced
- P/B picture
- MVMOD = “1 MV halfpel bilinear” or “1 MV bilinear”
- MVRANGE = 111b (max range)

The motion vectors are calculated by the differential motion vector and the predicted motion vector. See the following formula:

- mv = (dmv + pmv) smod(range)

When the Macro blocks are “1 MV half-pel bilinear” and the MVRANGE = 111b (max range), the ECD ***does not*** do the proper “smod” operation after calculating the Motion vector.

The error occurs only when the motion vectors are very long. Since most of the practical encoders have limitations on the search range, such motion vectors are seldom generated.

Workaround: For P and B frames when the MVRANGE = 111b immediately after the ECD task, check whether the MB type is “1 MV haflpel bilinear”.

When this condition is met, do the Motion vector calculations with "DSP" and update the ECD MB data in both the ECD auxiliary and ECD work buffers.

This advisory will be fixed in a future *major* silicon revision.

Advisory 1.1.14 *HDVICP – VC1 Encode: ECD Encodes Incorrect Bitstream When CBP = 0*

Revision(s) Affected: 1.1 and earlier

Details: This low impact issue is rare and occurs only when ***all*** the following conditions are met:

- Encode
- Advanced Profile
- Progressive/Interlaced
- P/B picture

The ECD does not encode the proper bitstream when all the following conditions are met:

1. the MB is coded with only 1 MV
2. there is no Motion vector (i.e., MV = 0)
3. there is no CBP

The above combination can be coded as a skip MB by the encoder. Hence this issue does not effect the encoder functionality.

Workaround: A MB which satisfies the above conditions can be treated as a skip MB. For the ECD to encode the MB as a skip MB, the DSP or ARM has to set the conditional skip flag. When there is no motion vector (i.e., MV is zero), it is recommended that the conditional skip flag be set so that the ECD can encode the corresponding MB as a skip when the CBP is zero.

This advisory will be fixed in a future *major* silicon revision.

Advisory 1.1.15 *HDVICP – VC1 Encode: ECD Encodes Incorrect Bitstream When Block is Less Than 8 x 8*

Revision(s) Affected: 1.1 and earlier

Details: This low impact issue is rare and occurs only when ***all*** the following conditions are met:

- Encode
- Simple/Main/Advanced Profile
- Progressive/Interlaced
- P/B picture

The ECD does not encode the proper bitstream when all the blocks of an MB are less than 8 x 8 partitions.

Workaround: It is recommended that < 8 x 8 partitions ***not*** be used. This limitation should not degrade the quality by much in the D1 or HD encode.

This advisory will be fixed in a future *major* silicon revision.

Advisory 1.1.17 — *HDVICP – H.264 Decode: Constrained Intra-Prediction –Affects High Profile Streams*

Revision(s) Affected: 1.1 and earlier

Details: This low impact issue is rare and occurs only when ***all*** the following conditions are met:

- H.264 Universal Decode
- High Profile
- MBAFF
- Constrained_intra_pred_flag = 1*
- A frame MB pair has a field MB pair as its left neighbor (see Figure 6).
- The top MB of the left neighbor is "Inter" and the bottom MB is "Intra"
- The bottom MB of the Frame pair is Intra 8x8 (shown as the shaded area in Figure 6)
- The Block 2 of the Intra 8x8 is coded with one of the following modes:
  - Intra_8x8_Vertical
  - Intra_8x8_DC
  - Intra_8x8_Diagonal_Down_Left
  - Intra_8x8_Vertical_Left
- In this case, the intra-prediction of "blk 2" is incorrect, and this may cause the intra-prediction of "blk 3" to also be incorrect. The error of this MB can propagate to the following MBs of the current and future pictures.

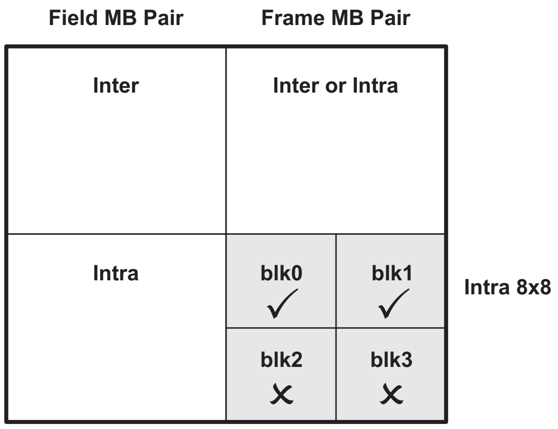


Figure 6. Conditions for the Advisory to Surface Example

The * constrained_intra_pred_flag equal to 1 specifies that the constrained intra prediction mode. For this case, the prediction of macroblocks coded using Intra macroblock prediction modes only uses residual data and decoded samples from I or SI macroblock types. (For more details, see Section 7.4.2.2, *Picture Parameter Set RBSP Semantics* of the H.264 standard document [*Advanced Video Coding for Generic Audiovisual Services, Series H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS, Infrastructure of Audiovisual Servicies – Coding of Moving Video*, ITU-T H.264 (03/2005)]).

Reason for this Advisory:

For calculating the intra prediction pixels of block2, the correct availability of pixels is:

- Upper pixels are available
- Left pixels are not available
- Upper left pixel (red) is available (see Figure 7)

The Calc module makes a mistake regarding the availability – it assumes that availability of the upper left pixel, which is NOT available, is the same as the left pixels.

As a result, the sample filtering before calculating the intra 8x8 prediction for block2 is wrong (For more details, see Section 8.3.2.2.1, *Reference Sample Filtering Process for Intra_8x8 Sample Prediction* of the H.264 standard document).

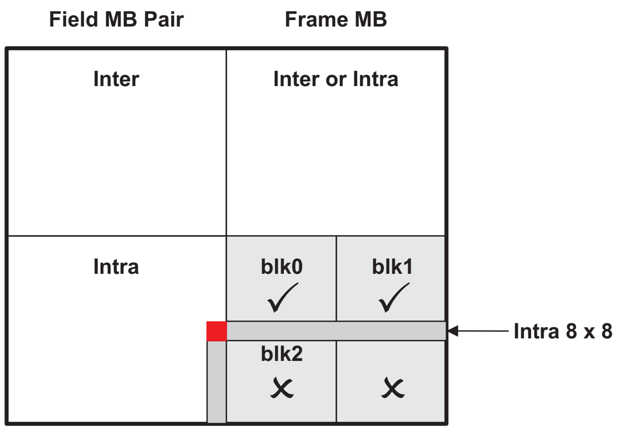


Figure 7. Availability of the Neighbor Pixels Example

Artifacts due to this Advisory:

- The luma pixels for the current Intra 8x8 MB are wrong
- If the following MBs are intra referring to this MB, the error is propagated spatially within this picture
- If the following pictures refer to this MB, the error is propagated temporally to the following pictures
- No effect on Chroma pixels as the upper left pixel is not used in Chroma prediction.

The probability that a real-time high-profile encoder will encode a macro block which satisfies all the above conditions w.r.t field/frame and inter/intra neighbors is very low.

Workaround: None. A software workaround is currently under investigation.

This advisory will be fixed in a future *major* silicon revision.

Advisory 1.1.18 *HDVICP – H.264 Decode: 128 Weight Value Weighted Prediction Not Supported*

Revision(s) Affected: 1.1 and earlier

Details: This low impact issue is rare and occurs only when ***all*** the following conditions are met:

- H.264 Universal Decode
- Advanced Profile
- Interlaced
- B picture
- MV RANGE of Anchor frame is bigger than current B picture
- MB type is forward or backward not direct or bi-Pred

When the weighted prediction mode flag is on, depending upon implicit or explicit mode selection, the following parameters are either derived (in implicit mode) or parsed from the bitstream (in explicit mode):

- Weight scalars W_0 and W_1 [W_1 is present only for B-type partitions/macroblocks]
- Offset values O_0 and O_1 [O_1 is present only during bi-prediction]
- logWD

At the beginning of every weighted prediction partition the MC – IP block needs to be initialized with these weighted prediction-specific parameters. The specific parameters for the weighted prediction are set as follows (For more details, see Section 8.4.2.3, *Weighted Sample Prediction Process* of the H.264 standard document.):

```
LWD = logWD  (single/bi-prediction mode)

OFS = O0 or O1        (single-prediction mode)
OFS = (O0 + O1 + 1) >> 1    (bi-prediction mode)
W0 = W0
W1 = W1
```

The weight scalars can have values from -127 to +128, with following joint constraint imposed: -128 ≤ W0 + W1 ≤ ((logWD = = 7) ? 127 : 128).

In both the implicit and explicit cases, there will be instances when a weight factor is equal to 128. The MC module has only a signed 8-bit field for a weight factor, -128 to 127; thus, when weight values take on the value of 128, the MC IP when initialized directly results in the wrong output.

This error occurs only when the weights take a value of 128. Such extreme weights are seldom used by encoders. Most of the cases are handled by the S/W Workaround which results in zero or negligible visual differences.

Workaround: **S/W Workaround:**

In P-picture, in the case where a weight factor is equal to 128, the equation of deriving the inter prediction is simplified as:
pred = (128 * pred_L0 + 128) / 256 + o0 = (pred_fwd + 1) / 2 + o0.

Note: that if the weight factor is equal to 128, the denominator is always 256 (For more details, see Section 8.4.2.3.2, Weighted Sample Prediction Process of the H.264 standard document). Then, the restriction of the signed 8-bit field can be overwritten by setting the logWD = 0 and W0 = 1.

In B-picture, there are two cases — 1. implicit mode and 2. explicit mode, during which weight factor can be equal to 128.

Case 1 (implicit mode): when W0 = 128, the W1 will be equal to -64 [see equation (8-278) of the H.264 standard document] and the logWD = 5; therefore,
pred = (128 * pred_L0 - 64 * pred_L1 + 32) / 64 + (o0 + o1 + 1) >> 1 is equivalent to
pred = (64 * pred_L0 - 32 * pred_L1 + 16) / 32 + (o0 + o1 + 1) >> 1

The restriction of the signed 8-bit field can be overwritten by setting the logWD = 4, W0 = 64 and W1 = -32.

Case 2 (explicit mode): when W0 = 128, W1 can vary from -128 to -1 while the logWD is restricted to 7. Let's assume that W1 is an even number e.g., when w0 = 128, w1 = 2N when N can range from -1 to -64 then,
pred = (128 * pred_L0 + 2N * pred_L1 + 128) / 256 + (o0 + o1 + 1) >> 1
= (64 * pred_L0 + N * pred_L1 + 64) / 128 + (o0 + o1 + 1) >> 1

The restriction of the signed 8-bit field can be overwritten by setting w0 = 64 and w1 = N. Otherwise if W1 is an odd number i.e., w1 = 2N – 1, where N can vary from 0 to -63, this simplification cannot be applied. For this particular case, the weight factor must be set to 127 (instead of 128), to minimize the drift in the pixel values. This same also applies when w1 = 128.

The signed 8-bit field restriction on a weight value in the MC IP affects the conformance of the bitstream in only one case and that case is when: explicit mode is used, the logWD is signaled as 7, and one of the weight values is present in the bitstream, while the other weight value is not present in the bit-stream hence it will be assigned the default value which is pow(2, logWD) = 128.

This advisory will be fixed in a future *major* silicon revision.

4 Silicon Revision 1.0 Usage Notes and Known Design Exceptions to Functional Specifications

This section describes the usage notes and advisories that apply to silicon revision 1.0 of the TMS320DM6467 device.

4.1 Usage Notes for Silicon Revision 1.0

Usage notes highlight and describe particular situations where the device's behavior may not match presumed or documented behavior. This may include behaviors that affect device performance or functional correctness. These usage notes will be incorporated into future documentation updates for the device (such as the device-specific data sheet), and the behaviors they describe will not be altered in future silicon revisions.

Silicon revision 1.0 applicable usage notes have been found on a later silicon revision. For more details, see Section 2.1, *Usage Notes for Silicon Revision 3.0*.

4.2 Silicon Revision 1.0 Known Design Exceptions to Functional Specifications

Some silicon revision 1.0 applicable advisories have been found on a later silicon revision. For more details, see Section 3.2, *Silicon Revision 1.1 Known Design Exceptions to Functional Specifications* and see Section 2.2, *Silicon Revision 3.0 Known Design Exceptions to Functional Specifications*.

Table 7. Silicon Revision 1.0 Advisory List

Title	Page
Advisory 1.0.1 —EMIFA I/F: NAND Flash Boot Mode Does Not Work	44

Advisory 1.0.1 *EMIFA I/F: NAND Flash Boot Mode Does Not Work*

Revision(s) Affected 1.0

Details The NAND Flash boot mode ***does not*** work due to the NAND busy signal being sampled by the EMIFA I/F too early in the address phase (before the NAND Flash starts driving the busy signal). As a result, the boot code prematurely determines the NAND Flash is ready for the next phase (the data phase) and starts reading the data when the NAND Flash is not ready yet.

Workaround(s) Use the SPI or I2C boot mode to load the User Boot Loader (UBL) into the on-chip memory (ARM TCM). The UBL will then load the user-application code from the NAND Flash to the on-chip or off-chip memory as appropriate.

The NAND boot loader loads the UBL from the NAND Flash to the ARM TCM and expects the UBL to load the rest of the user-application code. The only difference between the workaround and the on-chip NAND boot loader (NAND Flash boot mode) is the usage of serial EEPROM (SPI or I2C) to load the UBL.

This advisory will be fixed in the next silicon revision.

Revision History

This silicon errata revision history highlights the technical changes made to the SPRZ251F revision to make it an SPRZ251G revision.

Scope: Applicable updates relating to the TMS320DM6467 devices have been incorporated.

DM6467 Revision History

SEE	ADDITIONS/MODIFICATIONS/DELETIONS
Section 1	*Introduction*: • Updated/Changed "This document describes ..." paragraph to support device-specific TMS320DM646**7** versus DM646x
Section 1.2	*Revision Identification*: • Updated/Changed Figure 1, *Example, Device Revision Codes for TMS320DM6467 (ZUT Package)* • Updated/Changed the "*Silicon revision is identified by a ...*" paragraph • Updated/Changed the Silicon Revision 3.0 row in Table 1, *TMS320DM6467 Device Revision Codes*

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DLP® Products	www.dlp.com	Communications and Telecom	www.ti.com/communications
DSP	dsp.ti.com	Computers and Peripherals	www.ti.com/computers
Clocks and Timers	www.ti.com/clocks	Consumer Electronics	www.ti.com/consumer-apps
Interface	interface.ti.com	Energy	www.ti.com/energy
Logic	logic.ti.com	Industrial	www.ti.com/industrial
Power Mgmt	power.ti.com	Medical	www.ti.com/medical
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
RFID	www.ti-rfid.com	Space, Avionics & Defense	www.ti.com/space-avionics-defense
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Video and Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless-apps