ANALOG DEVICES
Digital Signal Processing Division

ADSP-2199x Anomaly List for Revisions up to 1.1

Created, 18th November 2009

These anomalies represent the currently known differences between revisions of ADSP-2199x devices and the functionality specified in the ADSP-2199x data sheets and ADSP-2199x DSP Core and Peripherals Manuals. A revision number with the form "-x.x" is branded on all parts.

The table below represents the known anomalies for all devices of the ADSP-2199x family.

Changes from the last version of this document (21st September 2006):

1. The sole purpose of this Anomaly List revision is to clarify this document's Silicon Revision (Rev) applicability. This Anomaly List applies to Silicon Rev 0.0 and 1.1 for the ADSP-21990, and Silicon Rev 0.0 for both ADSP-21991 and ADSP-21992. There is no technical content change from previous September, 2006 Anomaly List publication.

Anomaly List:

Description:
1. SPORT Active low, late frame sync failure
2. DMA Write with Pipeline Stall
3. Peripheral DMA with EMI DMA Writes
4. Real time update of PWM period
5 SPORT generates TFS (Transmit Frame Sync) one clock cycle earlier than expected when configured for data-dependent and early frame sync mode
6. SPORT fails to check for the frame sync in unframed mode after it is disabled and re-enabled
7. RFS and TFS guard cycles after SPORT enable
8. Serial clock is not gated by SPORT enable bit
9. Changing the serial port clock frequency on the fly
10. Processor's PLL may not lock or work at frequencies lower than 40 MHz
11. SPORT Multichannel mode [TDM] bit shift and loss of channel
12. Emulation logic problem with DMA

1. SPORT Active low, late frame sync failure:

   When the SPORT is configured in active low, late frame sync mode, the data will not get framed (T/RFSx may not be asserted through out the data transfer) for both transmitting and receiving.

   Workaround:
   Do not use active low, late frame sync mode. Use active high, late frame sync mode with an external inverter on frame sync.

2. DMA Write with Pipeline Stall

When a memDMA, ADC DMA or SPORT DMA is active, a DMA failure may occur if a pipeline stall coincides with DMA write activity. As a result, all DMA activity will be halted.
Pipeline stalls could happen for the following reasons:

a. Core IO read/writes (e.g. "AX0 = IO(0x300);")
b. Direct Core accesses to EMI - especially for reads.

The failure is independent of frequency.
The SPI Port will not trigger this problem.

Workaround:
If DMA activity cannot be avoided, steps must be taken to ensure pipeline stalls due to accesses initiated by the core do not occur at the same time as the DMA transactions.

- Do not perform I/O READ or WRITE transactions while memDMA , ADC DMA or SPORT is performing DMA WRITES to internal memory.
- Do not perform direct core external memory accesses while memDMA, ADC DMA or SPORT is performing DMA WRITES to internal memory.

Given that during Descriptor DMA the Core explicitly writes the ownership bit to give Ownership to the DMA peripheral, the core can monitor the Ownership bit (bit 15) in the DMA Configuration word in internal memory before doing any IO or external memory access. In other words, once the core hands over the DMA Ownership to the memDMA, ADC DMA or SPORT DMA, all further IO read/writes and external memory accesses should be restricted in software until the ownership is returned to the core (until DMA is completed). This means the core cannot do any IO accesses or external memory accesses when memDMA, ADC DMA or SPORT DMA is going on. It can still perform all the DSP computations and load/store from the internal memory.
This workaround only applies to Descriptor based DMA. There is no workaround for Autobuffer DMA.

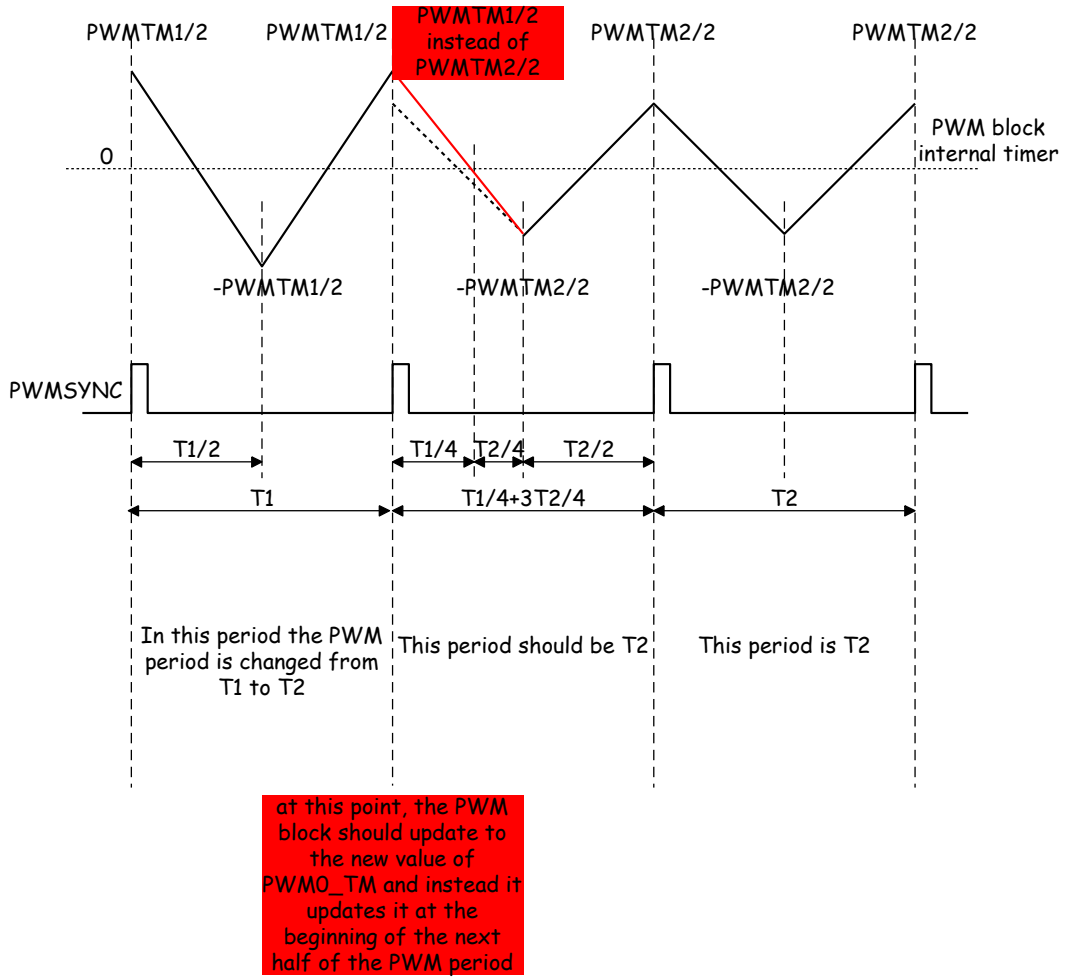
3. Peripheral DMA with EMI DMA Write

When two or more instances of memDMA/ADCDMA/SPORT DMA are performing WRITES to internal memory and an external memory DMA WRITE access is initiated by any other peripheral DMA, all DMA transactions may halt.
The anomaly only occurs when two or more instances of memDMA/ ADCDMA/SPORT DMA are performing WRITES to internal memory and a 3rd peripheral DMA attempts to perform a WRITE to external memory space. For example, the SPI could trigger the problem if they are performing DMA WRITES to external memory, and both memDMA and SPORT DMA are providing DMA WRITES to internal memory.

Workaround:
1. Do not perform any external memory DMA WRITES while two or more instances of memDMA/ ADCDMA/SPORT DMA are performing WRITES to internal memory.

4. Real time update of PWM period

When the PWM period is changed in real time (i.e. a new value is written in the register PWM0_TM), the first half of the new PWM period after the change is not as intended (please see figure below). After this first half, the following PWM periods have the right value. The anomaly happens in both single update and double update mode.



Workarounds:

1. For the first PWM period after modifying PWM0_TM register, impose the duty cycles PWMCHA, PWMCHB, PWMCHC knowing that this period is T1/4+3*T2/4 (in single update mode) or T1/4+T2/4 (in double update mode).
2. Disable and re-enable the PWM block. If both BIT 1 and BIT0 of PWM0_CTRL are cleared (0 is written over) and then immediately set back to 1, the PWM block changes instantaneously to the new PWM period. The drawbacks are: the current PWM period is abruptly interrupted and a new PWM period with the new value is started; a new PWMSYNC pulse is immediately generated, i.e. a supplementary PWM interrupt is furnished; the PWM outputs are disabled for 12 CCLK periods (in case HCLK=CCLK/2), the time to access two times the PWM0_CTRL register.

5. SPORT fails to check for the frame sync in unframed mode after it is disabled and re-enabled

The SPORT if configured for external frame sync, unframed mode, begins transmitting or receiving data without waiting for the frame sync after the SPORT is disabled and re-enabled.

The SPORT configuration would look like this. External frame sync (ITFS = 0 in SPx_TCR for transmit SPORT or IRFS = 0 in SPx_RCR for receive SPORT) and no frame sync required mode i.e. unframed mode (TFSR = 0/RFSR = 0).

The expected behavior is that the SPORT should check for the frame sync after it is disabled and re-enabled before it starts receiving or transmitting the data. But due to this anomaly the SPORT does not wait for the new frame sync after it is disabled and re-enabled.

This anomaly is not seen in the case of framed mode.

Workaround:

This is a hardware workaround to synchronize re-enabling of SPORT with the external frame sync. The external frame sync signal should be fed as input to a programmable flag of the processor which in turn triggers an interrupt. Inside the programmable flag interrupt service routine (ISR), the serial port should be re-enabled. Refer the steps given below:

- Sequence to disable the SPORT inside the SPORT ISR
    - Clear the SPORT-DMA interrupt
    - Disable the SPORT-DMA [In case DMA is used]
    - Disable the SPORT
    - Flush the DMA FIFO [In case DMA is used]
    - Re-configure the next DMA parameters [In case DMA is used]
    - Enable the SPORT DMA [In case DMA is used]
    - Enable the Programmable Flag (PFx) interrupt
- Steps inside the Programmable Flag (PFx) ISR
    - Clear the PFx interrupt
    - Disable the PFx interrupt
    - Enable the SPORT

6. SPORT generates TFS (Transmit Frame Sync) one clock cycle earlier than expected when configured for data-dependent and early frame sync mode.

Description:
The serial port generates TFS one serial clock cycle earlier than expected when configured for data-dependent and early frame sync mode. This anomaly occurs in core mode of SPORT data transfer. The failure occurs when the SPORT TX buffer is being loaded with a new data value while the SPORT TX shift register is in the process of shifting out the last bit of the previous word. The failure does NOT occur if:

A] The frame sync is configured to be data independent.

OR

B] The frame sync is configured to be late frame sync.

OR

C] DMA mode of data transfer is used.

When the SPORT operates in DMA mode, the failure will not be seen if the SPORT is configured for data dependent and early frame sync option.

Workarounds:

1. If none of the above modes ("A" or "B" or "C") can be used in a system, the alternative is to configure SPORT interrupt to high priority and enable interrupt nesting. This will ensure that the subsequent data word gets written in to TX buffer well in advance such that it does not coincide with the last bit being driven out of the TX shift register.

2. Wait until the complete word is transmitted out before writing the subsequent word in TX buffer. One of the following two approaches can be implemented to ensure that the last bit has been shifted out of the TX shift register.

a] The processor executes "nop;" instructions in a loop to wait for a fixed period of time before writing new data to TX buffer. The number of processor core cycles can be calculated based on SPORT serial bit clock frequency and the word length. For slower serial port clock frequency, executing several "nop;" instructions will affect processors performance. In this case approach "b" can be used.

b] Use timer interrupt to ensure that the last data bit has been shifted out of TX shift register. The main application code should initialize the timer parameters and the timer should be enabled inside the SPORT ISR before executing "rti;" instruction. Inside the timer interrupt, subsequent data can be loaded in to TX buffer.

Either of the above two workarounds would ensure that the load of transmit buffer does not coincide with the last bit of the data word being driven out of the TX shift register and would therefore avoid this anomaly.

7. RFS/TFS guard cycles after SPORT enable

Description:
To ensure proper functionality, when the SPORT Frame Sync (RFS or TFS) is provided externally, the Frame Sync must not become active within the first 6 SCLK cycles following the SPORT enable operation.

Workarounds:
1. Ensure that the externally generated SPORT frame sync [RFS or TFS] appears at least 6 SCLK cycles after the SPORT is enabled.

2. If the system cannot ensure "1" mentioned above, follow the steps below to make sure that the SPORT is enabled at least 6 SCLK cycles before the frame sync appears.

a] Limit the duration of the RFS/TFS pulse to one SCLK cycle.

b] Route the externally generated RFS/TFS signal to any spare flag pin of the processor, in addition to connecting it to the RFS/TFS pin of SPORT. Assuming active high frame sync, the Flag pin should be configured as a falling edge sensitive external interrupt.

c] Inside the interrupt service routine, disable the interrupt and enable SPORT. Note that this workaround will function correctly only if the word length is greater than 6.

8. Serial clock is not gated by SPORT enable bit

If the serial clock TCLK or RCLK is generated internally, the SPORT clock is generated regardless whether the SPORT is enabled or not. The serial clock divider, which is updated only when the SPORT is enabled, can produce unwanted clock pulses with the former SPORT frequency settings generated until the enable bit is set.

For example, after power on reset, SPORT SCLK is by default programmed for HCLK/2. When SPORT control register is written a value which configures ICLK = 1, the SPORT would still generate SCLK at HCLK/2 even if the SPORT is not enabled.

For external devices interfaced with SPORT, this would not be a problem if such devices take frame sync generated by the processor. This is because the frame sync stops when the SPORT is disabled even though the serial clock continues.

Workaround:

Configure the SPORT SCLK as external when not in use. To configure the serial clock as external clock, the following sequence should be followed:

a] Disable the serial port.
b] Clear the ICLK bit.

Note that the above sequence needs to be carried out in two separate instructions. The first instruction would disable the SPORT and the second instruction would clear the ICLK bit configuring SPORT clock as an external clock and therefore the clock would stop. Trying to clear ICLK bit in the same instruction as disabling the SPORT does not update the ICLK bit to "0".

9. Changing the serial port clock frequency on the fly

The effect of SPORT clock frequency change on-the-fly when the SPORT is enabled is incorrectly documented in the ADSP-2199x Mixed Signal DSP controller Hardware Reference manual rev. 0 - page 8-19.

Revised Description:
If the serial port is configured for an internal serial clock (ICLK = 1 in SPx_TCR / SPx_RCR registers) and the value of the serial clock frequency is changed (by writing into the SPx_TSCKDIV / SPx_RSCKDIV registers) when the SPORT is enabled, the change in TCLK/RCLK frequency takes effect immediately without waiting for the following frame sync (TFS / RFS).

Workaround:

In most of the systems this should not pose any problem because the change in frame sync period would also happen during the same time.

To ensure that the SPORT clock frequency change is synchronized with the occurrence of frame sync, the frame sync can be connected to the programmable flag. The following sequence is must be followed:

a] The application needs to change the SPORT clock frequency on the fly.
b] Enable the Programmable flag interrupt.
c] Inside the Flag interrupt service routine, disable the programmable flag interrupt.
d] Change the serial clock [TCLK/RCLK] frequency.

10. Processor's PLL may not lock or work at frequencies lower than 40 MHz

Description:
Processor's PLL may not lock or work when configured to generate a core clock frequency of less than 40MHz. This behavior is not observed at higher core clock frequencies.

Workaround:
For core clock frequencies lower than 40MHz, it is recommended to use the PLL by-pass mode.

11. SPORT Multichannel mode [TDM] bit shift and loss of channel

Description:
When the SPORT is configured for multi-channel [TDM] mode in either DMA or core mode following

issues are noticed:

- One cycle delay is introduced between the frame sync and data transmission for any programmed MFD. The data transmission starts with MFD + one cycle delay after the frame sync is de-asserted. This delay is noticed from the transmitter as well as the receiver side and therefore, if "0x8000" is transmitted in slot 1 and "0xa5a5" in slot 2, the words would be received as "0x2000" in slot 1 and "0x2969" in slot 2.
- The data frame will be shifted one bit all through the channel. This happens only during the first data frame transmission.
- The last data channel will be lost. i.e. If the transmitter is configured for 16 channels, only 15 channels will be transmitted and the 16^{th} channel will be lost. The loss of channel is seen only for MFD < 2. For MFD value of 2 or greater than 2, loss of channel is NOT noticed.
- In case of continuous transmission, the last channel of the first data frame will be transmitted as the first channel in the subsequent frame. No bit shift and channel loss is observed from the second transmission. Again, this is seen for MFD < 2. For MFD value of 2 or greater than 2, this behavior of channel shift is NOT noticed.

Workaround:

Case I: MFD < 2:

The following sequence of operations should be followed:

a] Program the transmitter and receiver for N+1 channels, say N is the required number of channels.

b] Ignore the first TDM data frame received.

c] Receive the second data frame and ignore the first data channel of each frame. This can be done by disabling the first channel of the received stream.

Case II: MFD >= 2:

Ignore the first TDM data frame.

12. Emulation logic problem with DMA

Description:

Let us assume that an autobuffer based DMA or a descriptor based DMA is in progress, and the execution of the program is stopped either due to a breakpoint or due to a halt (Shift + F5) command in VisualDSP++. After this, if any attempt is made to execute the program further, either by single stepping into the code (F11) or by running the code (F5), the VisualDSP ++ debugger displays an "Unknown" status at the bottom of the debugger window. VisualDSP ++ hangs and it is not possible to continue single stepping or execution of the code. This can be observed with any peripheral DMA.

If VisualDSP ++ is not stopped, the core execution and DMA continue with no problem.

Workaround:

This problem has been fixed in VisualDSP++3.5 March 2004 update. In the VisualDSP++ debugger window, go to settings -> target options and do NOT check on "Enable hardware single step".

VisualDSP++ will not get into the "unknown state" on halt and single-step/run actions during a DMA transfer.