Media Platforms and Services Group

ADSP-2191/95/96 Anomaly List for Revision 1.0

Created 25/04/2006

The table below represents the known differences between revision 1.0 ADSP-2191/95/96, and the functionality specified in the ADSP-2191/95/96 data sheets and the ADSP-219x/ADSP-2191 DSP Hardware Reference manual.
A revision number with the form "- x.x" is branded on all parts.

The left hand side of the table lists the type of anomaly and the numbers on the top of the table refer to the particular revision of the silicon. For a complete description of the anomaly or documentation update, please refer to the subsequent pages.

Changes from the last version of this document (1/11/2005):

1. Added anomaly 24 “SPORT generates TFS (Transmit Frame Sync) one clock cycle earlier than expected when configured for data-dependent and early frame sync mode”.

2. Re-phrased and updated anomaly 19 “SPORT fails to check for the frame sync in unframed mode after it is disabled and re-enabled”.

Anomaly List:

	0.2	1.0
1. UART THRE bit latency	A	A
2. UART Boot transmits ‘K’	A	A
3. SPORT Active low, late frame sync failure	A	A
4. Maximum HCLK is incorrectly documented	D	D
5. DMA Write with Pipeline Stall	A	X
6. Peripheral DMA with EMI DMA Writes	A	X
7. RFS/TFS guard cycles after SPORT enable	D	D
8. SPORT Multi-channel Mode starting and stopping	A	A
9. Serial clock is not gated by SPORT enable bit	A	A
10. SPORT MCM bit shift	A	X
11. DTx are driven immediately following SPORT Enable	D	D
12. SPORT Current Channel Status Indicator	X	A
13. SPORT MCM data transmission latency	X	A
14. DMA data loss when HCLK = CCLK	X	A
15. SPORT receive DMA bus priorities are incorrect	A	X
16. Emulation logic problem with DMA	A	A
17. Host port DMA read issue	A	A
18. SPI multiple slave mode with CPHA = 0	A	A

19. SPORT fails to check for the frame sync in unframed mode after it is disabled and re-enabled	A	A
20. SPORT may report a false transmit underflow status	A	A
21. Changing the serial port clock frequency on the fly	D	D
22. Setting the multiplier select pins to a 1x multiply factor	A	A
23. Host port autobuffer based DMA read issue	A	A
24. SPORT generates TFS (Transmit Frame Sync) one clock cycle earlier than expected when configured for data-dependent and early frame sync mode	A	A
Key: A = anomaly exists in revision, X = anomaly does not exist in revision, D = functionality has been incorrectly documented		

1. UART THRE bit latency:

There is a latency issue with the Transmit Holding Register (THR, I/O Page 0x05, I/O Address 0x00) of the UART which can cause a transmit data overrun situation to occur. In polling mode, the THR Empty (THRE) Bit within the Line Status Register (LSR, I/O Page 0x05, I/O Address 0x05) will be cleared too late. This can force a subsequent polling operation to fail.

Workaround:

The Transmitter Holding Register Empty bit (THRE, bit 5 of the Line Status Register, I/O Page 0x05, I/O Address 0x08) can be polled in software to acknowledge that the previous character loaded into the UART has been sent; at this point, the Transmit Holding Register can be loaded with the new character.
The following code is required to allow error-free transmissions via the UART by polling the THRE bit of the Line Status Register:

```
#define <def2191.h>
#define THRE 0x20

iopg = UART_Controller_Page;

wait_tx_rdy:
ar = IO(LSR);                   /* Check Line Status Register          */
ar = ar AND THRE;               /* Test THRE bit                       */
if eq jump wait_tx_rdy;         /* Wait until THR register shows empty */

IO(THR) = char_to_xmit;          /* Load THR data register             */

wait_thre:
ar = IO(LSR);                    /* Check the Line Status Register     */
ar = ar AND THRE;               /* Test THRE bit                       */
if ne jump wait_thre;           /* Wait until THR register shows FULL  */
```

2. UART Boot transmits ‘K’:

As documented in the ADSP-2191 Hardware Reference Manual, the Boot ROM expects external device transmits the byte “0xAA” to allow the timer to capture the pulse width of the device in order to determine the baud rate. Once the baud rate has been determined, the ADSP-2191’s UART will transmit the words “0x4F” and “0x4B” corresponding to “OK” in ASCII. However, as an impact of Anomaly#1 (UART

THRE bit latency), and therefore depending on Core Clock and UART Bit Rate, the ASCII character 'O' will not be transmitted in most cases.

Workaround:
The boot utility needs to accept both the 'OK' and a single 'K' as a valid reply from the DSP.

3. 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 in this revision of silicon. Use active high, late frame sync mode with an external inverter on frame sync.

4. Maximum HCLK is incorrectly documented:
The ADSP-219x/2191 DSP Hardware Reference Manual, First Edition (July 2001), incorrectly documents the maximum peripheral clock rate (HCLK) as 100MHz.

Revised Description:
The maximum peripheral clock rate is 80MHz. All functionality that is dependent on HCLK is still valid assuming a maximum HCLK of 80MHz.

5. DMA with Pipeline Stall
When a memDMA or SPORT DMA is active, a DMA failure may occur if a pipeline stall coincides with DMA 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.
UART, SPI, and Host Port peripherals will not trigger this problem.

Workaround:
1) 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 or SPORT is performing DMA READS and WRITES to internal memory
- Do not perform direct core external memory accesses while memDMA or SPORT is performing DMA READS and 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 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 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.

6. Peripheral DMA with EMI DMA Write

When two or instances of memDMA/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/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 HPI, UART, or 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/SPORT DMA are performing WRITES to internal memory.

7. RFS/TFS guard cycles after SPORT enable

To ensure proper functionality, when any 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 command.

Revised Description:

Enable the SPORT first and start RFS/TFS generation afterward (at least six cycles later). Otherwise you need to synchronize the DSP with the RFS/TFS generator accordingly, either at system level or by the following precautions:

a) Limit the duration of the RFS/TFS pulse to one SCLK cycle.
b) Route the RFS/TFS signal to any spare flag pin, in addition to the RFS/TFS pin.
c) Software delays the SPORT enable command until a trailing RFS/TFS edge is detected on that flag pin.

8. SPORT Multichannel Mode starting and stopping

If using the SPORTs in multichannel mode, the data will be corrupted if the SPORTs are first enabled, then disabled, and re-enabled again.

Workaround:

The following sequence provides correct data transmission if enabling, disabling, and re-enabling the SPORTs is necessary:

Disable SPORT -> Disable MCM operation -> SET FLSH bit -> wait a while -> CLR FLSH bit -> Stop DMA

This sequence must be completed before the next frame sync comes in. The SPORTs must be stopped on completion of a DMA transfer. If they are stopped midway through a multichannel mode DMA transfer, then on restart you will get unpredictable results. Both RX and TX are stopped separately. TX is stopped in the TX interrupt routine when it completes transmitting. RX is stopped in the RX interrupt routine when it completes receiving. The disabling of MCM operation must be done in the RX interrupt routine (it is not to be disabled in the TX interrupt routine).

9. Serial clock is not gated by SPORT enable bit

If one of the serial clocks, TCLK or RCLK, is generated internally, the clock is generated regardless whether the SPORT module 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. If the SPORT is enabled for the first time after reset, the observed

frequency is SCLK/2 since the clock-divide registers reset all to zero. Even if the ICLK bit is set at the same like the SPORT enable bit, a set of unwanted clock pulses is generated due to the SPORT enable latency. Note also that these leading clock pulses should not cause any issues if the SPORT is disabled and re-enabled again with the SCKDIV settings.

Workaround:
To disable the internally generated clock, first clear the enable bit. In a subsequent write operation, also clear the ICLK bit. Connect devices that tolerate the incorrect leading clock pulses. Note that most of devices ignore these pulses anyways, since no valid frame sync is generated during the critical period. If the connected device cannot tolerate incorrect clock pulse, the clock signal still can be gated and synchronized by off-chip logic controlled by a general-purpose flag pin during initial SPORT setup and run-time reconfigurations.

10. SPORT MCM BIT SHIFT
If the SPORT is configured for DMA MCM data unpacked mode, a problem may occur where the MSB/LSB of one channel gets shifted to the next channel. For example, transmit 0xff00 on slot 1 and 0x0010 on slot 2, you will receive 0x7f00 on slot 1 and 0x8010 on slot 2. The bit is shifted one whole slot not just along one bit. This happens with both 8 and 16bit data transfers.

Workaround:
This can be solved by increasing the TX FIFO pre-fetch distance or by using data packed mode.

11. DTx are driven immediately following SPORT Enable
Revision A of the ADSP-2191 Datasheet, Revision PrA of the ADSP-2195 Datasheet, and Revision PrA of the ADSP-2196 Datasheet incorrectly document the SPORT DTx active timing.

Revised Description:
Following SPORT enable in default mode (non-multichannel), the DT0, DT1, and DT2 become active before the TFS0, TFS1, or TFS2 occurs. This applies for both internal and external TFSx.

12. SPORT Current Channel Status Indicator
If the SPORT is configured in non-DMA multichannel mode, a problem will occur where the CHNL status bits are not updated when the SPORT transitions from the last channel of the frame to the first channel of the next frame. This means that the last channel number of the previous frame is reported twice in the CHNL status and the first channel of the next frame is not reported in the CHNL status.
For example: After enabling the SPORT for non-DMA multichannel mode, data transmission will start on the first frame. The CHNL bits in the SPx_STATR register will increment starting from the first frame.
In the case of a 16 channel multichannel scenario, the CHNL bits will be updated as follows:
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,15,1,2,3,4,5,6,7,8,....

Workaround:
Detection of the current channel can be made by monitoring the CHNL status in software. The first time the CHNL status reflects the last channel (in the case of a 16 channel scenario, first time encounter 0x0F), software should interpret it as the last channel of the current frame. The second time the CHNL status reflects the last channel, software should interpret it as the starting channel of the next frame. This can be done in the ISR as shown below.

Example code:

```
sport_tx_isr:
        IOPG = SPORT0_Controller_Page;
        ax0 = IO(SP0_STATR);            // Fetch CHNL Counter
        sr1 = ax0;
        sr = lshift sr1 by -4 (lo);
```

```
        ax0 = 0x007F;

        ar = sr0 and ax0;
        ay0 = buffer_length - 1;
        ax1 = ar;
        ar = ar - ay0;                  // Check CHNL counter
        if eq jump test_channel;        // If CHNL counter = last channel jump
                m0 = ax1;
                jump continue_tx;
test_channel:
        ar = dm(test_flag);             // Test CHNL to determine if first or second time
        none = pass ar;
        if eq jump last_tx_word;        // If first time, jump
                ax0 = 0;
                dm(test) = ax0;         // Toggle test_flag
                m0 = 0;                 // Set modifier to transmit word 0
                jump continue_tx;
last_tx_word:
        ax0 = 1;
        dm(test_flag) = ax0;            // Toggle test_flag
        m0 = ax1;                       // Set modifier to transmit last word
continue_tx:
        ax0 = dm(m0,i0);
        io(SP0_TX) = ax0;               // Load Data to TX
        RTI;
```

13. SPORT MCM Data Transmission Latency

If the SPORT is configured in multichannel mode, after enabling the SPORT, a problem may occur where data transmission will not begin until the start of the second frame. For example, after enabling the SPORT for multichannel mode, data transmission will not start on the first frame. Data transmission will begin at the start of the second frame.

Workaround:

There is no known workaround for this anomaly.

14. DMA data loss when HCLK = CCLK

When HCLK equals CCLK, DMA data loss may occur when multiple active DMA channels are accessing internal memory at the same time.

Workaround:

When HCLK equals CCLK, ensure that the DMA word counts for all SPORT and MEMDMA transfers are even multiples of 4 and that the total number of active DMA channels accessing internal memory are 4 or less.

15. SPORT receive DMA bus priorities are incorrect

On silicon revision 0.2, the DMA bus priorities do not match those published in the ADSP-219x/2191 DSP Hardware Reference Manual, First Edition (July 2001). The Sport Receive channels are given the lowest priority instead of the highest.

For silicon revision 0.2, the DMA bus priorities are as shown in the table below:

Highest
Sport0 TX - Transmit Channel
Sport1 TX - Transmit Channel

Sport2 TX	- Transmit Channel
SPI 0	- Read or write
SPI 1	- Read or write
UART RX	- Receive channel
UART TX	- Transmit channel
HPI	- Read or write
MemDMA	- Read Channel
MemDMA	- Write Channel
Sport2 RX	- Receive Channel
Sport0 RX	- Receive Channel
Sport1 RX	- Receive Channel

Lowest

Workaround:
There is no known workaround for this anomaly. On silicon revision 1.0, the DMA bus priorities have been fixed to match those published in the ADSP-219x/2191 DSP Hardware Reference Manual, First Edition (July 2001).

16. Emulation logic problem with DMA
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. This problem exists on rev.0.2 as well as rev.1.0 of the silicon. However, due to the DMA with pipeline stall anomaly (anomaly #5 in this document) in rev.0.2 silicon, this problem could get offset in rev.0.2 silicon.
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.

17. Host port DMA read issue
When a host processor does a DMA read from ADSP-2191/95/96 via the host port interface, the first word read is erroneously read as 0. This problem occurs for autobuffer based DMA read as well as descriptor based DMA read.

Workaround:
1) Since only the first word read is dropped, the starting address for the host port DMA read can be one less than the intended starting address and the count should be 1 more than the required value.
2) If the starting address cannot be changed, the host processor can do a direct access read for the first word only after the end of the host port DMA read transfer. In this case, count will be the same as the desired value.

18. SPI multiple slave mode with CPHA = 0
When the SPI is a master and CPHA = 0, the expected behavior is that the programmable flags that are used as slave select pins are asserted/de-asserted automatically by hardware. This functionality works fine when the SPI master is connected to a single SPI slave. However, when the SPI is a master that transmits

data to multiple SPI slaves, the programmable flags that are used as slave select pins may not function properly in the CPHA = 0 mode.

Workaround:
This problem does not occur when CPHA = 1. In this mode, software control is needed to control the slave select pins. The programmable pins should be asserted in software before data transmission and they should be de-asserted after the end of data transfer. The CPHA mode can be selected in the SPI Control (SPICTLx) register.

19. 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 ie, 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 [In case DMA is used]
    - 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 in side the Programmable Flag (PFx) ISR
    - Clear the PFx interrupt
    - Disable the PFx interrupt
    - Enable the SPORT

20. SPORT may report a false transmit underflow status
When a SPORT is configured to transmit data in the DMA mode with an external clock (ICLK = 0 in SPx_TCR) and external frame sync (ITFS = 0 in SPx_TCR), the SPORT Status register (SPx_STATR) may falsely report a transmit underflow error (TUVF = 1). The status register would show a transmit underflow enabled (TUVF = 1) along with a simultaneous transmit buffer full (TXS = 1) status. This false status is reported only once, just after enabling the SPORT.

Workaround:
1) If this status is reported just after enabling the SPORT, ignore the status.
2) Configure the serial port to use an internal transmit frame sync (ITFS = 1 in SPx_TCR) instead of external.

21. Changing the serial port clock frequency on the fly
The timing details of the serial port on-the-fly clock divider change mentioned in the ADSP-219x/2191 DSP Hardware Reference manual rev. 1.1, August 2003 are incorrectly documented.

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), the change in TCLK/RCLK frequency takes effect without waiting for the next frame sync (TFS / RFS). However the transmitted / received data is not corrupted and the new data bit rate is same as the new serial clock frequency.

22. Setting the multiplier select pins to a 1x multiply factor
If the input clock frequency (CLKIN) is lesser than 32 MHz and if the multiplier select (MSEL6:0) pins are set to a 1x multiply factor, the PLL may not get locked properly and the DSP core could hang. This behavior is not applicable to higher CLKIN frequencies and higher multiplication factors.

Workaround:
Use the bypass mode by pulling the BYPASS pin high instead of using the 1x multiply factor in the above case.

23. Host port autobuffer based DMA read issue
When a host processor does an autobuffer based DMA read from ADSP-2191/95/96 via the host port interface, the last word read is incorrect. The last word read is the first word in the array. For example, let us assume that the DMA size is 16. The host would read only first 15 words and then again read the first data in the array in place of the 16th word. It will never read the 16th word. This problem is not seen in case of a descriptor based DMA.

Workaround:
1) If the host is setup to read N words from the DSP using the autobuffer based DMA, setup the host to read (N+1) words instead. The last word read can be ignored.
2) Please note that if anomaly #17 is also observed, the starting address for the host port DMA read can be one less than the intended starting address (as a workaround to anomaly 17) and the count should be 2 more than the required value. For example, let us assume that the host is reading the following array (dataBuf[8]) from the DSP:

Address	Value
0x80FF	X
dataBuf: 0x8100	0x0001
0x8101	0x0002
0x8102	0x0003
0x8103	0x0004
0x8104	0x0005
0x8005	0x0006
0x8006	0x0007
0x8107	0x0008
0x8108	Y

When the workaround is not used and when anomalies 17 and23 are observed, the host would read the 8 words from the array dataBuf as:
{0x0000 (anomaly #17),
 0x0002,
 0x0003,
 0x0004,
 0x0005,
 0x0006,
 0x0007,
 0x0001 (anomaly #23)}

The workaround would be that the host does the DMA read starting from DSP address (dataBuf-1) instead of starting address dataBuf. Moreover, instead of reading 8 words, the host would read 10 words, by increasing the actual DMA count by 2. When this workaround is used, the host would read as:

{0x0000(anomaly #17),
 0x0001,
 0x0002,
 0x0003,
 0x0004,
 0x0005,
 0x0006,
 0x0007,
 0x0008,
 0x0001(anomaly #23)}

The host can then ignore the first word and the last word read from the DSP.

24. 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.