MAX32630 ERRATA SHEET

Revision A2 Errata

The errata listed below describe situations where components of this revision perform differently than expected or differently than described in the data sheet. Maxim Integrated Products, Inc., intends to correct these errata when the opportunity to redesign the product presents itself.

This errata sheet only applies to components of this revision. These components are branded on the topside of the package with a six-digit code in the form yywwRR, where yy and ww are two-digit numbers representing the year and work week of manufacture, respectively, and RR is the revision of the component. To obtain an errata sheet on other die revisions, visit the Maxim website at www.maximintegrated.com/errata.

1) PMU WRITE DESCRIPTOR MASK DOES NOT OPERATE AS EXPECTED

Description:
The PMU descriptor mask is expected to operate as:

(NEW_VALUE & MASK) | (ORIG_DATA & ~MASK)

but operates as:

NEW_DATA | (ORIG_DATA & ~MASK).

Workaround:
AND NEW_VALUE with MASK before writing to the WrData register.

2) PMU ONLY SUPPORTS CHANNELS 0–4

Description:
PMU channel 5 does not operate as intended. With the provided workaround, PMU channels 0–4 are available.

Workaround:
The workaround consists of three steps:

1. Set clock gating to ALWAYS-ON. This software workaround is implemented in the latest version of the SDK/API. No workaround for this step is necessary if the supplied drivers are used.

2. PMU channel 5 must be set in a loop, jumping back to itself. This software workaround is implemented in the latest version of the SDK/API. No workaround for this step is necessary if the supplied drivers are used.

3. MOVE and TRANSFER commands must have the STOP bit set.

3) USB HARDWARE CANNOT DETECT SUBSEQUENT SUSPEND EVENTS AFTER INITIAL SUSPEND/RESUME CYCLE

Description:
After a USB bus reset, the USB peripheral properly detects only the first USB bus suspend condition. Subsequent suspend conditions are not detected until a bus reset occurs.

Workaround:
Upon detection of resume signaling from the host, set the SIGRWU bit to cause a remote wakeup. Signaling for a remote wakeup is the same bus state as resume, but is shorter in duration. Therefore, this remote wakeup is benign and is not seen by the host. The remote wakeup should only be issued upon detection of resume signaling. If the bus activity is a bus reset, the remote wakeup should not be issued.

This workaround resolves the issue, and the device again can detect a suspend condition. Example software is available that implements this workaround.

4) FLASH MEMORY PAGE PROTECTION FEATURE DOES NOT OPERATE AS EXPECTED

Description:
The flash memory page protection control registers do not operate as expected.

Workaround:
None. Do not modify the page protection control registers from their default state.

5) SPI MASTER DOES NOT OPERATE AS EXPECTED IN MODE 1, 2, OR 3

Description:
The SPI master does not operate as intended in mode 1, 2, or 3.

Workaround:
None.

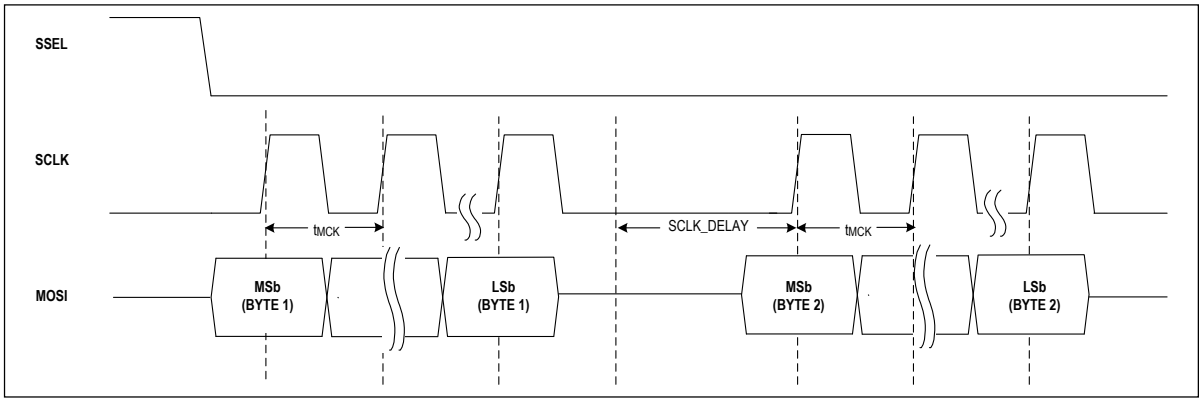
6) SPI MASTER INTERBYTE DELAY IN MODE 0

Description:
The SPI master remains low one or more SCLK cycles after transmitting 8 bits of data, resulting in an interbyte delay when transmitting back-to-back bytes. This erratum does not affect the integrity of the synchronous SPI bus because no data is clocked in or out by the SPI slave during the delay time SCLK is asserted low.

In single mode, the delay occurs after SDIO[0] has transmitted all 8 bits.

In dual mode, the delay occurs after SDIO[1:0] have each transmitted 4 bits.

In quad mode, the delay occurs after SDIO[3:0] have each transmitted 2 bits.



The length of the delay is a function of the SPI clock frequency as shown in the following table. The SPI master must be operated in sample mode with a maximum frequency of 24MHz.

SCLK OUTPUT FREQUENCY (f_{MCK})	SCLK_DELAY
SPIM peripheral clock/2	2 x $t_{(SPI\ peripheral\ clock)}$
SPIM peripheral clock/4	3 x $t_{(SPI\ peripheral\ clock)}$
SPIM peripheral clock/8	5 x $t_{(SPI\ peripheral\ clock)}$
SPIM peripheral clock/16	9 x $t_{(SPI\ peripheral\ clock)}$
SPIM peripheral clock/32	17 x $t_{(SPI\ peripheral\ clock)}$
SPIM peripheral clock/64	33 x $t_{(SPI\ peripheral\ clock)}$
SPIM peripheral clock/128	65 x $t_{(SPI\ peripheral\ clock)}$
SPIM peripheral clock/256	129 x $t_{(SPI\ peripheral\ clock)}$

Workaround:
None.

7) SPI SLAVE DOES NOT OPERATE AS EXPECTED

Description:
The SPI slave peripheral does not operate as expected.

Workaround:
None. Do not use any instance of the SPI slave peripheral.

8) P1.7 DOES NOT OPERATE AS EXPECTED WHEN 32kHz OUTPUT MODE ON P1.7 IS ENABLED

Description:
Enabling the 32kHz clock output mode on P1.7 configures P1.6 as an input with an internal pullup. The primary and secondary functions, pulse train output and timer input are unavailable in this configuration. P1.6 remains in this state until the 32kHz clock output mode on P1.7 is disabled. The internal pullup configuration increases power consumption in low power modes.

Workaround:
Do not use P1.6 except as a GPIO input while the 32kHz clock output mode is enabled.

9) GPIO_FREE_P1.[7:6] ARE INCORRECT WHEN 32kHz OUTPUT MODE ON P1.7 IS ENABLED

Description:
When the 32kHz output mode on P1.7 is enabled, GPIO_FREE_P1.7 GPIO = 1 and GPIO_FREE_P1.6 = 0. Instead the bits should be GPIO_FREE_P1.7 GPIO = 0 and GPIO_FREE_P1.6 = 1.

Workaround:
Software should recognize the bits are inverted when reading GPIO_FREE_P1.[7:6].

10) READS OF IOMAN_SPIM2_REQ[31:16] RETURN IOMAN_SPIM2_REQ[30:15]

Description:
Reads of IOMAN_SPIM2_REQ[31:16] instead return IOMAN_SPIM2_REQ[30:15]. As a result the acknowledge bits in IOMAN_SPIM2_ACK[31:16] do not align with the request bits in IOMAN_SPIM2_REQ[31:16].

Workaround:
To read the correct value for IOMAN_SPIM2_REQ[31:16], implement the following (pseudocode) procedure:

```
scratch = IOMAN_SPIM2_REQ & 0x7FFF8000/mask off all bits except 30:15

scratch = (scratch << 1)        // scratch now contains the correct values in bits 31:16
```

Revision History

REVISION NUMBER	REVISION DATE	DESCRIPTION	PAGES CHANGED
0	8/16	Initial release	—