TMS470R1A384

TMS470 Microcontroller

Silicon Errata

Silicon Revision E

SPNZ148B
January 2007

REVISION HISTORY

This silicon errata revision history highlights the technical changes made to the SPNZ148 for each revision. Page numbers refer to the specified revision.

REVISION	ADDITIONS/CHANGES/DELETIONS
SPNZ148	Initial release
SPNZ148A	Added ADM#11 through ADM#14; pages 6−7 Added DEV#42; page 8 Added I2C#1 through I2C#34; pages 13−20 Added Known Exceptions to Timing Specifications; page 25
SPNZ148B	Corrected page headers; all pages

Contents

1 Known Design Marginality/Exceptions to Functional Specifications

The following is a list of advisories on modules in this version of silicon, errata matrix version 8.89. Documentation may differ from the user guide or data sheet. The advisory reference number is shown first (i.e.; ADM#8), followed by a description and any known workarounds. The reference numbers may not always be sequential for this device.

Modules include the following:

Multi-buffered analog-to-digital converter (ADM)

Clock control module (CCM)

Device specific (DEV)

Direct memory access controller (DMA)

Expansion bus module (EBM)

Flash wrapper (FW)

General-purpose input/output (GIO)

High-end timer (HET)

Inter-integrated circuit (I2C)

Real-time interrupt (RTI)

Standard CAN controller (SCC)

Serial peripheral interface (SPI)

Zero-pin phase-locked loop (ZPLL)

Advisory ADM#8	*Stopping and Starting ADC When Conversions Are Ongoing*

Description: When used in FIFO mode, if the A to D module is disabled or if the channel select registers are cleared while conversions are still ongoing, the operation will be unpredictable when the module is restarted.

Workaround: Stop the ADC by clearing ADCR1(5).
Restart the ADC by setting ADCR1(5) again.
Configure all groups to be in single conversion mode.
Configure one channel to be converted in all three groups and start the conversions.
Wait for these conversions to end by polling the ”conversion end” flags in the ADSR register.
Clear the channel select registers for the three groups.
Continue with the desired configuration for the ADC.

Advisory ADM#9 — *Freeze Feature Error for Conversion Groups*

Description: When multiple conversion groups are being used and the ADC is used in the multi-buffered mode, the use of the freeze feature for conversion groups can lead to conversion results being written to the wrong FIFO. If a conversion group (say group A) is configured to be ”freezable”, and if there is a request for servicing another conversion group (say group B) while group A conversion is still ongoing, then the conversion result for the last channel converted in group A will be written to the FIFO for group B.

Workaround: Do not use the freeze ability for the conversion groups.
OR
For applications that must use the freeze ability, please use only the compatibility mode of the ADC.

Advisory ADM#10 — *Data Not Written to MibADC FIFO*

Description: In buffered mode, if the channel select register is written and if no other MibADC registers are accessed, the converted data does not get written to the FIFO but the threshold counter is updated upon each conversion. This occurs only when the ICLK/SYSCLK ratio is more than 2.

Workaround: Do a read operation from the same group input channel select register after writing it.

Advisory ADM#11 — *Disabling EN Bit Does Not Properly Stop and Restart the ADC*

Description: Disabling the ADC by clearing the ADC EN bit of the ADCR1 register does not reset some internal flags used to store the conversion statuses of the three groups, causing problems when stopping and restarting the ADC conversions in buffered mode.

Workaround: TBD

Advisory ADM#12 — *Clearing Channel Selection Register Does Not Clear FIFO Completely*

Description: Clearing the channel selection register does not clear FIFO completely under certain conditions.

Workaround: Depending upon the number of selected channels, the overhead on ”time” is in the increasing order with each workaround. Depending upon the requirement and criticality, any of them can be chosen.

1. Irrespective of whether a group is in continuous or single conversion mode, read out the corresponding group’s FIFO data until it’s empty, write all 0s to the Group Channel Select Register. After this, wait for the duration of one Channel Conversion completion including the sampling and the conversion time, before writing the next set of channels to the Channel Select Register.

2. Alternatively, if the group conversion is in continuous mode, and an application wants to change the Group Channel Select Register, then first change the mode of the group to single conversion. Read out all the conversion data from the FIFO, until the FIFO becomes empty. Write a single channel into the Channel Select Register. Wait until the Group Conversion End flag gets set. Write the next set of channels to the Group Channel Select Register.

3. If the group is in Single conversion mode, wait until the Group Conversion End flag is set, read out the FIFO data until it’s empty and then write the new set of Channels to the Group Channel Select Register.

Advisory ADM#13 — *Current Channel Conversion Does Not Complete When Suspend Mode Is Entered*

Description: Ongoing conversion is not completed on entering suspend mode when COS = 1.

Workaround: TBD

Advisory ADM#14 — *Calibration Results Are Written to Group FIFO*

Description: If calibration is started during a group conversion, the result of the calibration is written to that group FIFO until the end of group conversion.

Workaround: Do not do calibration during a group conversion. The documentation will be updated to reflect this requirement. (SPNU193, 9/2002)

Advisory CCM#1 — *ICLK Not 50% Duty Cycle*

Description: The ICLK signal output from the CCM is not a 50% duty cycle signal when the SYSCLK to ICLK divide ratio is odd. This affects the SCI and SPI modules and occurs when the divide ratio is 3 or above.

Workaround: None

Advisory DEV#42 — *5V Tolerant Pins Do Not Have Internal Clamp Diode to Positive Voltage*

Description: Pins with 100-μA internal pullups should be specified with an input current of between −200 μA to −100 μA.

Workaround: The documentation will be updated to reflect this requirement. (SPNU193, 9/2002)

Advisory DMA#4 — *BMSS=1 Mode Not Supported*

Description: DMA transfers in BMSS=1 mode will be corrupted due to a bug in the DMA state machine.

Workaround: BMSS=1 mode is no longer supported. Use BMSS=0. The documentation will be updated. (SPNU194, 11/2002)

Advisory DMA#15 — *CPU Reads of DMA Control Packet Memory*

Description: If the ARM7 CPU is reading the DMA control packet memory while the DMA is operating, the DMA control packet configuration word or the DMA control packet transfer count can be corrupted.

Workaround: The DMA SPD version 1:10 avoids this problem by keeping a copy of the DMA control packet configuration words in RAM. Using the latest version of this SPD will avoid the problem. Do not read the DMA control packet memory while the DMA is operating. Be careful to avoid instructions that perform a read-modify-write operation on the DMA control packet memory while the DMA is operating.

Advisory DMA#17 — *Reads of MPU Registers Corrupt Data*

Description: If the ARM7 CPU is reading a memory protection unit (MPU) register while the DMA is operating, the data read or written by the DMA can be corrupted.

Workaround: Avoid any reads of the MPU registers while the DMA is operating. CPU reads of the MPU registers while the DMA is operating are not supported. The documentation will be updated. (SPNU194, 11/2002)

Advisory DMA#19	*No Exception for DMA Access to Unmapped Memory on Expansion Bus*

Description: No reset or abort occurs when the source or destination address of the DMA is an unmapped memory area on the expansion bus.

Workaround: None.

Advisory DMA#20	*No Exception for DMA Access to Unmapped Memory on Expansion Bus*

Description: No reset or abort occurs when the source or destination address of the DMA is an unmapped memory area on the expansion bus.

Workaround: None. The documentation will be updated to clarify that memory bounds checking is not supported on DMA accesses to the expansion bus. (SPNU194, 11/2002)

Advisory DMA#21	*One Transfer With Zero Transfer Count*

Description: If a control packet is set up and enabled with a transfer count of zero, one DMA transfer occurs.

Workaround: Do not enable a DMA control packet with a transfer count of zero.

Advisory DMA#23	*DMA Stop Corrupts Command Buffer Memory*

Description: Using DMA Stop may corrupt the DMA command buffer memory.

Workaround: Use DMA Halt, not DMA Stop.

Advisory DMA#24	*DMA Writes to Read-Only Memory Do Not Generate an Illegal Address*

Description: When a particular region of memory is set as read only by the address decoder or the MPU, any write to that memory region should generate an illegal access. This works properly in the case of CPU writes, but DMA writes do not cause an illegal access. In both cases, writes to the RAM are blocked by blocking the chip selects.

Workaround: None

Advisory DMA#25 — *DMA Channel Switch Size Not Properly Documented*

Description: For DMA transfers on the expansion bus, the channel switch size is documented properly − that is, values of 0 to 15 give a switch size of 1 to 16. For transfers on the CPU bus, channel switch size of zero gives one transfer. Channel switch sizes 1 to15 give 1 to15 transfers.

Workaround: The documentation will be updated. (SPNU194, 11/2002)

Advisory DMA#26 — *Half-Word and Byte Writes to Unimplemented Bits Corrupt Register*

Description: Half-word or bytes to the high-order bytes of DMA Global Control register or the DMA Global Disable register will corrupt these registers.

Workaround: The documentation will be updated to warn users about this condition. There is no reason to write to these unimplemented bits. (SPNU194, 11/2002)

Advisory DMA#27 — *Data Chaining With More Than One Active Channel*

Description: When more than one DMA channel is active and data chaining is used on one or more channels, one extra transfer is done on a previously serviced DMA channel even if a new request comes on a different higher priority channel.

Workaround: Either do not use data chaining, or do not use more than one channel.

Advisory DMA#28 — *DMA Fails During Execution of the SWP Instruction*

Description: When a DMA transaction is supposed to happen during the CPU execution of an SWP instruction that accesses memory, the DMA transaction does not happen.

Workaround: Halt the DMA whenever the SWP instruction must be used.

Advisory DMA#29 — *DMA Corrupts PSA*

Description: If a DMA transaction occurs on the cycle before writing to the PSA enable bit to disable the PSA, the PSA will be corrupted.

Workaround: Halt the DMA before disabling PSA.

Advisory EBM#1 — *Only 22 Address Lines Supported*

Description: The specification indicates that the EBM module supports a 27-bit address when using the 8-bit access mode. However, the address lines 22 to 26 are tied off low, and so are not available for use. Therefore, only 22 address lines are supported.

Workaround: None

Advisory FW#3 — *Configuration Mode Required for Sleep or Standby*

Description: The configuration mode must be set to enter sleep or standby modes.

Workaround: The documentation will be updated to reflect this requirement. (SPNU213, 12/2002)

Advisory FW#13 — *Fails Initial Read of 0x0−0x7 in Pipeline Mode*

Description: Immediately after entering pipeline mode, a data read of location 0x04 immediately following a data read of location 0x0 will cause 0x04 to read as all 0s.

Workaround: Do a dummy data read of any location other than zero or four immediately after entering pipeline mode. The documentation will be updated to reflect this requirement. (SPNU213, 12/2002)

Advisory FW#15 — *No Wakeup From Powerdown in Pipeline Mode*

Description: On this device, flash banks in pipeline mode put into standby/sleep mode can not wake up by doing a normal read access or any other wake-up interrupt. Therefore, it is not possible to use the automatic power-down feature of banks that are not accessed for a given number of cycles.

Workaround: Do not use automatic power down of banks on this device during normal operation in pipeline mode.

Advisory FW#18	*No Read From Last Word in Pipeline Mode if Bank Is in Sleep or Standby*

Description: If not all banks are in sleep/standby (but at least one is), and an access to the last word of the last bank is performed in pipeline mode, a prefetch to a nonexisting bank that is in sleep/standby (because nonexisting banks power down signals are tied to existing bank power-down signals − normally bank0) will cause the CPU to hang.

Workaround: Last word of last bank should not be used.

Advisory GIO#1	*Reading the Interrupt Offset Registers*

Description: When either of the two interrupt offset registers are read, and a higher priority interrupt occurs in the same cycle, the interrupt pending flag for the higher-priority interrupt is wrongly cleared, but the offset for the lower-priority interrupt is read. As a result, the lower-priority interrupt will be serviced twice and the higher-priority interrupt will not be serviced at all.

Workaround: Do not read the interrupt offset register to identify the pending interrupt with the highest priority. Instead, read from the interrupt pending flag register and use bit tests to decode the pending interrupt with the highest priority by software. An additional write to the flag register is necessary to clear the pending interrupt flag.

Advisory HET#15	*Auto Read Clear Malfunction*

Description: The HET Auto Read Clear feature does not always work properly. Specifically, the data field of instruction X is not cleared if the following conditions a) and b) are true at the same time:

a) The 64-bit CPU read access happens exactly in the two HET time slots preceding the time slot Y in which instruction X is executed.

b) Instruction X just changes its data field (in time slot Y). (Example: Instruction X is an ECNT instruction, which just detected an edge.) The malfunction does not occur if the data field of instruction X does not change, since then b) is not true.

Workaround: See above.

Advisory HET#16 — *MCMP Causes a Constant Signal, Not PWM*

Description: MCMP causes a constant signal instead of a PWM, if both of the following conditions are met:

1. Consecutive compare match in every LRP for order=reg_ge_data (only when [data=0]).
2. The high resolution delay (in number of SYSCLK cycles) is equal to the time slot the MCM is executed.

Workaround: Replace each MCMP with a two instruction sequence: ECMP and MOV32.

Advisory HET#19 — *Reading the Interrupt Offset Registers*

Description: When either of the two interrupt offset registers are read, and a higher priority interrupt occurs in the same cycle, the interrupt pending flag for the higher-priority interrupt is wrongly cleared, but the offset for the lower-priority interrupt is read. As a result, the lower-priority interrupt will be serviced twice and the higher-priority interrupt will not be serviced at all.

Workaround: Do not read the interrupt offset register to identify the pending interrupt with the highest priority. Instead, read from the interrupt pending flag register and use bit tests to decode the pending interrupt with the highest priority by software. An additional write to the flag register is necessary to clear the pending interrupt flag.

Advisory I2C#1 — *Slave Cannot Detect Stop Condition*

Description: There is no way currently for the slave to detect, when the master has generated a stop condition. This is a potential bug and for a generic application, the I2C cannot be used as a slave in interrupt mode due to this limitation.

Workaround: TBD

Advisory I2C#2 — *Extra Transmit Interrupt by Slave*

Description: An extra transmit interrupt is generated by the slave transmitter in interrupt mode.

Workaround: Pad the message with a dummy byte whether in receive or transmit mode.
The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#3 — *Stop Condition Clears TXRDY*

Description: STOP condition clears TXRDY in slave non-DLB mode.

Workaround: TBD

Advisory I2C#4 — *Resetting the $\overline{IRS}$ Bit Does Not Clear the Bus Busy Bit in Master Mode*

Description: Resetting the $\overline{IRS}$ bit does not clear the bus busy (BB) bit in master mode. This occurs because of the assumption of it working even if the I2C is in software reset, however, it does not work when the I2C is in software reset. The BB bit is set by not only a start condition but also by a low state on SCL, so that the BB bit can show the correct state of the I2C bus as soon as the I2C goes out of reset.

Workaround: The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#5 — *Unexpected Behavior with 10-bit Addressing W-R Format*

Description: The I2C module cannot behave with 10-bit addressing W-R format as described in the Philips I2C Spec Rev 1.2 Section 14.2. The I2C sends a full address after a repeated start condition for reading. The I2C sends the slave address second byte every time it sends the slave address first byte.

Workaround: The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#6 — *ISCPSC Affects Real Dividing Value*

Description: I2CPSC affects real dividing value on rising edge of $\overline{IRS}$. The value of the I2CPSC is transferred to the internal register which controls the I2C clock frequency on rising edge of $\overline{IRS}$. As far as $\overline{IRS}$ is 0, the frequency of the I2C clock is not changed even if I2CPSC is updated.

Workaround: The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#7 — *Resetting During Transfer*

Description: Resetting the $\overline{\text{IRS}}$ bit during transfer can the hang the I2C bus.

Workaround: The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#8 — *Bus Busy Bit Not Cleared by Resetting the IRS Bit*

Description: The bus busy (BB) bit is not cleared by resetting the $\overline{\text{IRS}}$ bit.

Workaround: The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#9 — *Bus Fails to Go Busy in Master Transmitter Mode*

Description: If the I2C is configured in master transmitter mode and another master transmitter transfer is initiated immediately after the previous one, the start condition is not issued and the bus fails to go busy for the next transfer. The end of the first transfer is checked by polling for the bus busy bit. However the MST bit (for the master transmitter mode) does not get reset at the same instant when the bus busy bit goes free. As a result, if another master transmitter transfer is started before the MST bit for the previous transfer is reset, the MST bit for the second transfer gets reset (because of the first transfer resetting the MST bit). As a result, the I2C module fails to recognize itself as the master. In that case, when the start condition is issued the I2C module fails to get the bus busy.

Workaround: Wait for the MST bit (in addition to the bus busy bit) to get reset before starting another master transmitter transfer. The documentation will be updated to reflect this requirement.

Advisory I2C#10 — *Bit Count Bit Description*

Description: Clarification of the description of the bit count bits in the I2CMDR.

Workaround: The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#11 — *Module Appears to Hang When SCL Low*

Description: If an external I2C master is pulling SCL low, the I2C is programmed for master mode, and the Start has been issued while the input clock is still low; then no I2C activity will be observed and the module appears to be hung, requiring a reset to release the bus.

Workaround: TBD

Advisory I2C#12 — *ARDY Bit Set and Cleared Without Being Written To*

Description: The bit ARDY of I2C register I2CSTR is being set and then cleared with out being written to.

Workaround: TBD

Advisory I2C#13 — *I2C Write of I2CMDR Followed by Read Miscompare*

Description: An I2C write of I2CMDR is immediately followed by a read miscompare. A read of I2CMDR immediately following a write returns the data before the write. Further reading of I2CMDR produces the value written. This occurs when the STP stop bit is set during a master mode, receive, repeat mode transfer. The transfer is stopped at the end the next byte.

Workaround: The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#14 — *RSFULL Bit Set on First Data Byte After Start*

Description: In the I2CSTR register, the bit RSFULL is set on the first data byte after start while in master-receive repeat mode with extended address.

Workaround: The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#15	*I2C SDA Data Transition While SCL is High*

Description: I2C SDA data transition occurs while SCL is high. The SDA line is driven low while the SCL line was high during a data transmission. This is a I2C protocol fault which occurs in master, transmit, extended address, and repeat mode when the ICCLKL and ICCLKH registers were both programmed to 2 with the ICPSC = 2. The problem appears to be related to the clock divider registers = 2. These register settings are outside of the range for normal I2C rates and, therefore, there may need to be some limits specified for programming these registers.

Workaround: Do not use the I2C module under these conditions. The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#16	*RXRDY and TXRDY Not Cleared by Reading I2CIVR*

Description: The interrupt flag bits RXRDY and TXRDY are not cleared by reading I2CIVR.

Workaround: The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#17	*TXRDY Not Reset by Writing 1*

Description: The I2C register I2CSTR bit TXRDY is not reset by writing a 1 to it.

Workaround: The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#18	*I2CCLKH and I2CCLKL Not Functioning as Simple Clock Dividers*

Description: The I2C clock divider registers I2CCLKH and I2CCLKL are not functioning as simple clock dividers.

Workaround: The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#19	*Clearing I2CIVR Not Specified in Register Description*

Description: Clearing the I2CIVR is not clearly specified in the register description.

Workaround: The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#20	*Unused Bits Not Cleared in Bits Mode*

Description: In bits mode, unused bits are not cleared to 0 and bits from the old data in the shift register appear instead.

Workaround: TBD

Advisory I2C#21	*I2COAR/I2CSAR Register Description Update*

Description: Clarification must be added to the I2COAR/I2CSAR register description.

Workaround: The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#22	*Unexpected ICXEVT Event and ICXRDY Interrupt*

Description: Unexpected ICXEVT event and ICXRDY interrupt in master/XA/transmit/repeat mode.

Workaround: The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#23	*Contents of I2CDRR Not Lost at an Overrun*

Description: The contents of I2CDRR are not lost at an overrun.

Workaround: The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#25	*False ARDY Status*

Description: False ARDY=1 status occurs after starting in master, transmit, repeat, and FDF mode.

Workaround: The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#26 — *No NACK Status*

Description: No NACK status occurs when NACK received in slave, and transmit mode.

Workaround: The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#27 — *Peripheral Clock Frequency Range*

Description: The peripheral clock frequency range is documented incorrectly.

Workaround: The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#28 — *No NACK Status When Received in Master STB Mode*

Description: No NACK status occurs when NACK is received in master STB (start byte) mode.

Workaround: The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#29 — *Stop Not Cleared on NACK in STB Mode*

Description: STP (stop) is not cleared on NACK in STB (start byte) mode.

Workaround: The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#31 — *Slave Transmit With Extended Address Fails After Repeat Start*

Description: A slave transmit mode operation with extended address fails to ACK after the repeat start and first seven bits of address are resent. The conditions are slave-transmitter, 10-bit mode, ICMDR[2:0](BC) = 001b. The I2C does not work correctly with this configuration. The I2C regards repeated start condition following the second byte of the slave address as an illegal start condition and follows it as a normal start condition and starts receiving data as a new slave address. But the next first byte of the slave address is ”11110XX0”, which is illegal as a first byte address. It is valid only when it is the third address phase in 10-bit addressing mode. As a result, the I2C thinks that it is not addressed by the master module and goes back to idle state.

Workaround: The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#32 — *BB Bit Does Not Reflect I2C Bus Status Correctly*

Description: In all modes of operation, the BB bit does not reflect the I2C bus status correctly. This is a serious issue when the device is configured in multi-master mode.

Workaround: The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#33 — *AAS Bit Not Cleared by START Condition*

Description: In 10-bit addressing mode, the AAS bit is not cleared by repeated Start condition.

Workaround: The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory I2C#34 — *False Start Condition on Reset*

Description: The I2C will experience a false start condition if the module is reset while the bus is busy. As a result, the I2C block will begin clocking in data in the middle of a transmission and invalid data will be received.

Workaround: The documentation will be updated to reflect this requirement. (SPNU223, 2/2005)

Advisory RTI#3 — *Tap Interrupt When Clearing Counter*

Description: Write accesses to the RTICNTR register will clear the CNTR (21 bit counter), which will cause a Tap interrupt if the corresponding bit switches from a 1 to a 0.

Workaround: Disable the RTI prior to changing the RTICNTR value.

Advisory RTI#4 — *Tap Interrupt When Clearing Counter in Suspend Mode*

Description: Write accesses to the RTICNTR register will clear the CNTR (21 bit counter) which will cause a Tap interrupt if the corresponding bit switches from a 1 to a 0 when the suspend signal is asserted.

Workaround: This is the same problem as RTI#3, however, on the initial fix of RTI#3, the case where the suspend signal is asserted because of an emulator breakpoint was not considered. This problem occurs when the emulator has set a breakpoint on one of the instructions closely following the instruction which writes to the counter.

Advisory RTI#6 — *Asynchronous Clear of RTI Tap Flag Not Recommended*

Description: When using the oscillator to clock the RTI counter, any asynchronous clear of the RTI tap flag could cause an arbitration condition between the clear and the RTI module trying to set the flag. This will cause the tap flag to not get set and, hence, the Tap interrupt to not occur.

Workaround: Before attempting to clear the RTI tap flag, the RTI counter needs to be checked to make sure that the RTI module is not about to set the flag again.

Advisory SCC#4 — *CAN Does Not Perform Resynchronization As Expected*

Description: Due to the proposed update of the ISO-WD-16485 CAN Test specification (2001-05-31), the HCC/SCC on this device has a nonconformance to the Bosch CAN Specification and the ISO-11898 Standard as described below:

If the following conditions are met, the CAN does not perform a resynchronization as it is expected.

Conditions:

1. The node must be transmitter.
2. The node must transmit a dominant bit.
3. The dominant bit must be sampled back as recessive.
4. A recessive to dominant edge must be detected after the sample point.

But since the recessive sampling of the bit transmitted as dominant is an error anyway, an error frame will be transmitted at the beginning of the following bit.

Therefore, the effect of the nonconformance is a delay of this error frame. The maximum for this delay is five (max(SJW) + 1 Tq) time quanta.

Workaround: This nonconformance is classified as nonserious and does not have any impact on proper communication and interoperability with other nodes. See above description.

Advisory SCC#6 — *CANHRX Must be High During Self-Test*

Description: The CANSRX pin must be high during self-test.

Workaround: The CANSRX pin is usually driven high by the bus transceiver. As long as there is no bus activity during the self-test, this is not a problem. If there is nothing driving the CANHRX pin, it can be configured as a digital output and set high during the self-test.

Advisory SCC#7 — *Abort Acknowledge Bit Not Set After Transmission Request Reset*

Description:

After aborting a message using the Transmission Request Reset (TRR) register bit, there are some rare instances where the TRR bit will clear without setting the Abort Acknowledge (AA) bit.

In order for this rare condition to occur all the following three conditions must happen:

1. The current message has a message error or lost arbitration. This message does not need to have the same mailbox number as the following TRR bit mailbox.
2. The TRS bit of the same mailbox as the TRR mailbox must be set from either this current message, prior to the current message and still pending, or just set.
3. The TRR bit must be set in the exact ICLK cycle where the wrapper state machine is in IDLE for one cycle. (One ICLK before or after, and the condition will not occur). This IDLE state can occur just after the current message. It can also occur just a few ICLKs after setting the TRS bit of any mailbox after the current message (point 1 above).

If these conditions occur, then the TRR and TRS bits for the mailbox will clear t_{clr} ICLKs after the TRR bit is set where:

t_{clr} = ((16−mailbox_number)*2)+3 ICLK cycles

The TA and AA bits will not be set if this condition occurs. Normally, either the TA or AA bit sets after TRR bit goes to zero.

Workaround:

When this problem occurs, the TRR and TRS bits will clear within t_{clr} ICLK cycles. To check for this condition, first disable the interrupts. Check the TRR bits' t_{clr} ICLK cycles after setting the TRR bits to make sure that they are still set. A set TRR bit indicates the problem did not occur. If TRR is cleared, then maybe it was the normal end of a message and the TA or AA bits are set. Check both the TA and AA bits. If they are both zero, then the conditions did occur. Handle the condition like the interrupt service routine would, except that the AA bit does not need clearing now. If the TA or AA bit is set, then the normal interrupt routine will happen when the interrupt is reenabled.

Advisory SPI#1 — *Slave Baud Rate Setting*

Description:

When the SPI is operated in slave mode, the SPI clock must be configured to a baud rate as close to the master's baud rate as possible. If the baud rate is too slow, the enable signal will not be generated in time to keep the master from sending additional data. If the baud rate is too fast, the slave will capture the data before the last bit is shifted in.

Workaround:

The documentation will be updated to reflect this requirement. (SPNU195C, 7/2003)

Advisory SPI#2	*Clearing, Setting SPI EN Bit Does Not Clear Internal Flag*

Description: Clearing and then setting the SPIEN bit does not clear an internal flag that indicates there is valid data in the SPI data register. This could lead to an inadvertent overrun error. The software should do a dummy read of SPIBUF after setting the SPIEN bit to clear the internal flag.

Workaround: The documentation will be update to reflect this requirement. (SPNU195C, 7/2003)

Advisory ZPLL#1	*Clock Corruption When Changing Multiplier*

Description: All interrupt requests coming to the CIM module must be disabled when changing between multiply-by-4 and multiply-by-8.

Workaround: Disable the interrupt request at the peripheral source if possible.

2 Known Exceptions to Timing Specifications

Advisory *SPIn Master Mode Timing Parameters Correction at 125°C*

Description: At 125°C, the SPIn master mode timing parameter #4 (clock phase = 1) needs to be as follows:

			Current MIN Spec in ns	New MIN Spec at 125°C in ns
4	$t_{v(SIMO-SPCH)M}$	Valid time, SPInCLK high after SPInSIMO data valid (clock polarity = 0)	$0.5t_{c(SPC)M} - 10$	$0.5t_{c(SPC)M} - 12$

The current specification, $0.5t_{c(SPC)M} - 10$, is valid at 85C.

Advisory *EBM Timing Modifications*

Description: Based on characterization data, the expansion bus timing parameters are as follows:

NO.	PARAMETER	MIN	MAX	UNIT
2	$t_{d(COH-EBADV)}$		26	ns
3	$t_{h(COH-EBADIV)}$	(−8)		ns
4	$t_{d(COH-EBOE)}$		15	ns
5	$t_{h(COH-EBOEH)}$	2		ns
6	$t_{d(COL-EBWR)}$		17	ns
7	$t_{h(COL-EBWRH)}$	3		ns
8	$t_{su(EBRDATV-COH)}$	25		ns
9	$t_{h(COH-EBRDATIV)}$	(−6)		ns
10	$t_{d(COL-EBWDATV)}$		19	ns
11	$t_{h(COL-EBWDATIV)}$	(−12)		ns
12	$t_{d(COH-EBCSn)}$	6	20	ns
13	$t_{h(COH-EBCSnH)}$		21	ns
14	$t_{su(COH-EBHOLDL)}$	25		ns
15	$t_{su(COH-EBHOLDH)}$	25		ns