ERRATA SHEET

Date: **2009 March 12**
Document Release: **Version 2.0**
Device Affected: **LPC2104, LPC2104/00**

This errata sheet describes both the functional problems and any deviations from the electrical specifications known at the release date of this document.

Each deviation is assigned a number and its history is tracked in a table at the end of the document.

NXP
Semiconductors

Document revision history

Rev	Date	Description
2.0	2009 March 12	1. Document revision table was added
		2. Added Rev E

Identification:

LPC2104 devices typically have the following top-side marking:

> LPC2104xxx
>
> xxxxxxx
>
> xxYYWW R

LPC2104/00 devices typically have the following top-side marking:

> LPC2104xxx
>
> /00
>
> xxxxxxx
>
> xxYYWWR

The last letter in the last line (field 'R') will identify the device revision. This Errata Sheet covers the following revisions of the LPC2104 and LPC2104/00:

Revision Identifier (R)	Comment
'A'	Initial device revision
'B'	Second device revision
'C'	Third device revision
'D'	Fourth device revision
'E'	Fifth device revision

Field 'YY' states the year the device was manufactured. Field 'WW' states the week the device was manufactured during that year.

Errata Overview - Functional Problems

Functional Problem	Short Description	Device Revision the problem occurs in
IAP.1	No return from IAP erase/program call	A, B, C
SPI.1	Unintentional clearing of SPI interrupt flag	A, B, C, D
SPI.2	Incorrect shifting of data in slave mode at lower frequencies	A, B, C, D
VPBDIV.1	Incorrect read of VPBDIV	A, B, C, D
CORE.1	Incorrect load of the link register	A, B, C, D, E
Timer.1	Missed Interrupt Potential	A, B, C, D
PWM.1	Missed Interrupt Potential for Match Functionality	A, B, C, D
UART.1	Coinciding VPB read and hardware register update	A, B, C, D
Reset.1	Device does not power up correctly under certain internal conditions	A, B, C

Errata Overview - AC/DC Deviations

AC/DC Deviations	Short Description	Device Revision the deviation occurs in
$I_{pu}.1$	Different I_{pu} values	A, B, C, D, E

Functional Problems of LPC2104

IAP.1   **Flash memory programming interface timing problem**

Introduction: The Flash memory on the LPC2104 offers In-Application Programming (IAP) functionality. The IAP routines are part of the on-chip boot loader software, which controls the interface between the digital logic and the Flash memory. Please note that all programming methods (JTAG, ISP, IAP) use IAP calls.

Problem: Due to a timing problem in the interface between the Flash block and the digital logic the following problem may occur:

                 If the boot loader revision in the device is previous to V1.52 then in up to 10% of the devices the Flash memory interface at some point during an IAP programming or erase operation may never return from the IAP call. Please note that devices that pass the IAP programming are functional and do not suffer from any long-term reliability problems.

                 Devices with a date code prior to 0426 (manufactured before week 26 in 2004) are generally affected by this problem unless you receive devices with updated boot loader software from your distributor. Parts marked with date code 0426 or later are not affected by this problem. Please refer to page 2 of this document for details on how to identify the date code.

Workarounds:

                 1) The on-chip boot-loader software can be updated via ISP to correct this issue. The boot loader update files can be downloaded here:

                 http://www.nxp.com/files/products/standard/microcontrollers/utilities/lpc2000_bl_update.zip

                 The boot-loader version can be read out using the Philips Flash ISP Utility which can be found here:

                 http://www.nxp.com/files/products/standard/microcontrollers/utilities/lpc2000_flash_utility.zip

                 Both links can also be found on the product information page for this product under the Support & Tools section which can be found here:

                 http://www.nxp.com/pip/LPC2104.html

                 2) Limiting the external clock frequency to 12 MHz AND making sure the on-chip PLL is turned OFF while programming any part of the Flash memory reduces the likelihood of the occurrence significantly. During In-System-Programming the PLL is turned off by default.

SPI.1   **Unintentional clearing of SPI interrupt flag**

Introduction: The SPI interrupt flag is set by the SPI interface to generate an interrupt. It is cleared by writing a 1 to this bit.

Problem: A write to any register associated with the SPI peripheral will clear the SPI interrupt register.

work-around: Avoid writing to SPI registers while transmissions are in progress or while SPI interrupts are pending.

SPI.2   **Incorrect shifting of data in slave mode at lower frequencies**

Introduction: In slave mode, the SPI can set the clock phase (CPHA) to 0 or 1.

Problem: Consider the following conditions:

a. SPI is configured as a slave (with CPHA=0).

b. SPI is running at a low frequency.

In slave mode, the SPIF (SPI Transfer Complete Flag) bit is set on the last sampling edge of SCK. If CPHA is set to 0 then the last sampling edge of SCK would be the rising edge.

Under the above conditions, if the SPI Data Register (SPDR) is written to less than a half SCLK cycle after the SPIF bit is set (this would happen if the SPI frequency is low) then the SPDR will shift data one clock early for the next transfers.

Lowering the SPI frequency would increase the likelihood of the SPDR write happening in the first half SCK cycle of the last sampling clock.

Work-around: There are two possible workarounds:

1) Use CPHA=1.

2) If the data is shifted incorrectly when CPHA is set to 0 then delaying the write to SPDR after the half SCK cycle of the last sampling clock would resolve this issue.

VPBDIV.1   **Incorrect read of VPBDIV**

Introduction: The Peripheral Bus Divider (VPBDIV) divides the processor clock (CCLK) by one, two, or four. This is the clock that is provided to the peripheral bus.

Problem: Reading the VPBDIV register may return an incorrect value.

work-around: Performing two consecutive reads of the VPBDIV assures that the correct value is returned.

TIMER.1   **Missed Interrupt Potential**

Introduction: The Timers may be configured so that events such as Match and Capture, cause interrupts. Bits in the Interrupt Register (IR) indicate the source of the interrupt, whether from Capture or Match.

Problem: If more than one interrupt for multiple Match events using the same Timer are enabled, it is possible that one of the match interrupts may not be recognized. If this occurs no more interrupts from that specific match register will be recognized. This could happen in a scenario where the match events are very close to each other. This issue also affects the Capture functionality.

Specific details:

Suppose that two match events are very close to each other (Say Match0 and Match1). Also assume that the Match0 event occurs first. When the Match0 interrupt occurs the 0th bit of the Interrupt Register will be set. To exit the Interrupt Service Routine of Match0, this bit has to be cleared in the Interrupt Register. The clearing of this bit might be done by using the following statement:

T0_IR = 0x1;

It is possible that software will be writing a 1 to bit 0 of the Interrupt Register while a Match1 event occurs, meaning that hardware needs to set the bit 1 of the Interrupt Register. In this case, since hardware is accessing the register at the same time as software, bit 1 for Match1 never gets set, causing the interrupt to be missed.

In summary, while software is writing to the Interrupt Register, any Match or Capture event (which are configured to interrupt the core) occurring at the same time may result in the subsequent interrupt not being recognized.

Similarly for the Capture event, if a capture event occurs while a Match event is being is serviced then the Capture event might be missed if the software and hardware accesses coincide.

Affected features:

1. Interrupt on Match for Timer0/1.

2. Interrupt on Capture for Timer0/1.

3. These same features will be affected when using PWM.

Work-around: There is no clear workaround for this problem but some of the below mentioned solutions could work with some applications.

Possible workarounds for Match functionality:

1. If the application only needs two Match registers then distribute them between Timer 0 and Timer 1 to avoid this problem.

2. Stop the timer before accessing the Interrupt register for clearing the interrupt and then start timer again after the access is completed.

3. Polling for interrupt: Supposing that there are two Match events (Match X and Match Y). At the end of the Interrupt Service Routine (ISR) for Match X, compare the Timer Counter value with the Match Register Y value. If the Timer Counter value is more than the Match Register Y value then it is possible that this event might have been missed. In this case jump to the ISR directly and service Match event Y.

Possible workarounds for Capture functionality:

1. Try to spread the capture events between both timers if there are two capture events. If the application also has a match event then one of the capture events may suffer.

2. Polling for Capture: At the end of a Match interrupt ISR or Capture event ISR compare the previous Capture value with the current Capture value. If the Capture value has changed then the Capture event might have been missed. In this case, jump to the ISR directly and service the Capture event.

PWM.1   **Missed Interrupt Potential for the Match functionality. The description is same as above.**

Core.1   **Incorrect update of the Abort Link register in Thumb state**

Introduction: If the processor is in Thumb state and executing the code sequence STR, STMIA or PUSH followed by a PCrelative load, and the STR, STMIA or PUSH is aborted, the PC is saved to the abort link register.

Problem: In this situation the PC is saved to the abort link register in word resolution, instead of half-word resolution.

Conditions:

The processor must be in Thumb state, and the following sequence must occur:

<any instruction>

<STR, STMIA, PUSH> <---- data abort on this instruction

LDR rn, [pc,#offset]

In this case the PC is saved to the link register R14_abt in only word resolution, not half-word resolution. The effect is that the link register holds an address that could be #2 less than it should be, so any abort handler could return to one instruction earlier than intended.

Work around: In a system that does not use Thumb state, there will be no problem.

In a system that uses Thumb state but does not use data aborts, or does not try to use data aborts in a recoverable manner, there will be no problem.

Otherwise the workaround is to ensure that a STR, STMIA or PUSH cannot precede a PC-relative load. One method for this is to add a NOP before any PC-relative load instruction. However this is would have to be done manually.

UART.1   **Coinciding VPB read and hardware register update.**

Introduction: Reading the contents of the IIR,LSR and MSR registers will clear certain bits in the register.

1. Reading the IIR should clear the THRE status if THRE is the highest priority pending interrupt

(Only affects UART1).

2. Reading LSR should clear the OE/PE/FE/BI bits (affects both UART0 and UART1).

3. Reading MSR should clear the Delta DCD/Trailing Edge RI/Delta DSR/Delta CTS bits (Only affects UART1).

Problem: If hardware is setting one of these above bits while the software is reading the contents of the register the reading process clears all bits in the register including the bit that got set by hardware. The software reads the old value though and the bit that got set by hardware is lost.

Specific details:

Suppose IIR has a modem status interrupt while the other interrupts are inactive and software reads the IIR value (polling) while hardware sets the THRE interrupt then software will read the Modem Interrupt value while the THRE interrupt is cleared i.e the THRE interrupt is lost.

Suppose the LSR is all zeros and software is reading the register while hardware is generating a parity error then the parity error bit is cleared while the software reads the old value (all zeros) i.e. the parity error is lost.

Suppose MSR is all zeros and software is polling the value of the register while the value of CTS is changing then the change in CTS value should result in the Delta CTS bit getting set. Instead software will read all zeros and the Delta CTS bit in the MSR register will be cleared i.e. the Delta CTS status is lost.

Work-around:

IIR reading:

The IIR bug can be worked around by disabling the modem status interrupt effectively making THRE the lowest priority interrupt. The work-around does not work in software interrupt polling mode. Modem status has to be handled by software polling MSR.

Now there are two cases:

1. A THRE interrupt is pending, software responds to the interrupt by reading the IIR while another, higher priority interrupt is set (e.g. RDA). In this case software will read the THRE status although the status will not be cleared where it should have been. After handling the THRE and RDA interrupt another dummy THRE interrupt may occur, unless in the meantime software has filled THR. This is considered an error although not fatal.

2. A high priority interrupt is pending, software responds to the interrupt by reading the IIR register while a THRE interrupt is set. In this case, software will read the higher priority interrupt and the

THRE interrupt will be handled later. This behaviour is as expected.

LSR reading:

A work-around for this problem is to service the OE/PE/FE/BI condition before another character is received which will trigger an LSR update. So basically, service the interrupt in one-character time.

MSR reading:

The MSR bug can be worked-around by not using the Delta DCD/Trailing Edge RI/Delta DSR/Delta CTS bits in the MSR but instead use the DCD/TRI/DSR/CTS bits in the same register. To prevent, a transition from being missed software should poll the register's value at a sufficiently high rate.

Reset.1   Device does not power up correctly under certain internal conditions

Problem: If certain rare chip-internal conditions are met, the device will not start up correctly when executing a power-on reset. The crystal oscillator will be running but the device will not execute code.

Workaround: Apply a second (warm) reset pulse (without power-on cycle). The minimum time requirement between the first (unsuccessful) reset and the second reset is 4105 external oscillator clock cycles, which means that the assertion of the second reset should occur 4105 cycles or more after the deassertion of the first reset. For example, at 10 MHz, this is equal to 411 μs; at 20 MHz, this is equal to 206 μs. This can be achieved by using an external watchdog timer or by any other circuitry in the application that is able to assert a second reset pulse.

The root cause for this problem has been identified and is fixed from Revision D of this device onwards. This problem is also fixed in the LPC2104/00 versions of this device which have dedicated order numbers (LPC2104FBD48/00 (Package- LQFP48) and LPC2104FHN48/00 (Package-HVQFN48)).

AC/DC Deviations of the LPC2104

$I_{pu}.1$: The limits for pull-up current, I_{pu} (which applies to pins P0.22-P0.31) at DC input voltage 0V have been previously set to a minimum value of -25uA and a maximum value of -65uA. The minimum value of I_{pu} can be as low as -15uA and the maximum value can be as high as -85uA.