ERRATA SHEET

Date: **June 16, 2006**
Document Release: **Version 1.6**
Device Affected: **LPC2131**

This errata sheet describes both the functional deviations 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.

2006 June 16

Philips Semiconductors

PHILIPS

Identification:

The LPC2131 devices typically have the following top-side marking:

LPC2131xxx

xxxxxxx

xxYYWW R

The last letter in the third line (field 'R') will identify the device revision. This Errata Sheet covers the following revisions of the LPC2131:

Revision Identifier (R)	Comment
-	Initial device revision
'A'	Second device revision
'B'	Third Device revision
'C'	Fourth Device revision
'D'	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 History - Functional Problems

Functional Problem	Short Description	Errata occurs in device revision
Core.1	Incorrect update of the Abort link register	-, A, B, C, D
Timer.1	Missed Interrupt Potential	-, A, B
Timer.2	Timer Counter reset occurs on incorrect edge in counter mode	-, A, B, C, D
PWM.1	Missed Interrupt Potential for Match Functionality	-, A, B
Power Down.1	Restricted Vdd while device is in power down mode	A
BOD.1	BOD reset not functional	-
UART.1	Coinciding VPB read and hardware register update	-, A, B
MAM.1	Incorrect read of data from SRAM	-, A, B
SPI.1	Incorrect shifting of data in slave mode at lower frequencies	-, A, B
SSP.1	Initial data bits/clocks corrupted in SSP transmission	-, A, B, C, D
DC/DC.1	DC/DC Converter start-up issue	-, A, B, C

Errata History - Electrical and Timing Specification Deviations

AC/DC Deviations	Short Description	Errata occurs in device revision
ESD.1	ESD-HBM Stress issue	-, A, B

Functional Deviations of LPC2131

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 PC relative 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.

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 Timer.1.

Timer.2 In counter mode, the Timer Counter reset does not occur on the correct incoming edge

Introduction: Timer0 and Timer1 can be used in a counter mode. In this mode, the Timer Counter register can be incremented on rising, falling or both edges which occur on a selected CAP input pin.

This counter mode can be combined with the match functionality to provide additional features. One of the features would be to reset the Timer Counter register on a match. The same would also apply for Timer1.

Problem The Timer Counter reset does not trigger on the same incoming edge when the match takes place between the corresponding Match register and the Timer Counter register. The Timer Counter register will be reset only on the next incoming edge.

Work-around: There are two possible workarounds:

1. Combine the Timer Counter reset feature with the “interrupt on match” feature. The interrupt on match occurs on the correct incoming edge. In the ISR, the Timer Counter register can also be reset. This solution can only work if no edges are expected during the duration of the ISR.

2. In this solution, the “interrupt on match” feature is not used. Instead, the following specific initialization can achieve the counting operation:

a. Initialize the Timer Counter register to 0xFFFFFFFF.

b. If “n” edges have to be counted then initialize the corresponding Match register with value n-1. For instance, if 2 edges need to be counted then load the Match register with value 1

More details on the above example:

a. Edge 1- Timer overflows and Timer Counter (TC) is set to 0.

b. Edge 2- TC=1. Match takes place.

c. Edge 3- TC=0.

d. Edge 4- TC=1. Match takes place.

e. Edge 5- TC=0.

Power Down.1: Restricted Vdd while device is in power down mode.

Introduction: This device has two reduced power modes: idle mode and power down mode. In power down mode, the

is shut down and the chip receives no internal clocks.

Problem: If Vdd is below the specified value, a reset could occur, waking up the device and the saved values in SRAM will be re initialized according to a power-up procedure.

Work-around: The two possible workarounds would be:-

1. Avoid operating in power down mode.

2. If the device is operated in power down mode then Vdd should be 3.3V +10%/-5%.

Brown-out Detection(BOD).1: BOD reset does not get triggered when the voltage of Vdd falls below 2.6V

Introduction: The BOD monitors the Vdd in two stages. If the voltage falls below 2.9 V, the BOD asserts an interrupt signal to the Vectored Interrupt Controller. The second stage of the low voltage detection asserts Reset to inactivate the device when Vdd falls below 2.6V.

Problem: BOD reset does not get triggered when the voltage of Vdd falls below 2.6V.

Work-around: None.

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.

MAM.1 Incorrect read of data from SRAM after Reset and MAM is not enabled or partially enabled

Introduction: The Memory Accelerator Module (MAM) provides accelerated execution from the on-chip flash at higher frequencies.

Problem: If code is running from on-chip Flash, a write to an SRAM location followed by an immediate read from the same SRAM location corrupts the data been read. For instance, a stack push operation immediately followed by a stack pop operation

Work-around: User code should enable the MAM after Reset and before any RAM accesses; this means MAMTIM and MAMCR should be set as follows:

MAMTIM: For CPU clock frequencies slower than 20MHz, set MAMTIM to 0x01. For CPU clock frequencies between 20MHz and 40MHz, set MAMTIM to 0x02, and for values above 40MHz set

MAMTIM to 0x03.

MAMCR: Set MAMCR to 0x02 (MAM functions fully enabled)

MAMTIM should be written before MAMCR.

SPI.1 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 upcoming 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.

SSP.1 Initial data bits/clocks of the SSP transmission are shorter than subsequent pulses at higher frequencies

Introduction: The SSP is a Synchronous Serial Port (SSP) controller capable of operation on a SPI, 4-wire SSI or a Microwire bus. The SSP can operate at a maximum speed of 30MHz and it referred to as SPI1 in the device documentation.

Problem: At high SSP frequencies, it is found that the first four pulses are shorter than the subsequent pulses.

At 30MHz, the first pulse can be expected to be approximately 10ns shorter and the second pulse around 5ns shorter. The remaining two pulses are around 2ns shorter than subsequent pulses.

At 25MHz, the length of the first pulse would be around 7ns shorter. The subsequent three pulses are around 2ns shorter.

At 20MHz only the first pulse is affected and it is around 2ns shorter. All subsequent pulses are fine.

The deviation of the initial data bits/clocks will decrease as the SSP frequency decreases.

Work-around: None.

DC/DC.1 DC/DC converter start-up issue

Introduction The device operating voltage range is 3.0V to 3.6V and it is an internal DC/DC converter that provides 1.8V to the ARM7 Core.

Problem: If during a power-on reset the voltage on Vdd takes longer than 200ms to ramp from below 0.8V to above 2.0V, the chip-internal DC/DC converter might not start up correctly. If this happens, the crystal oscillator will not be running, resulting in no code execution. As an example, having a Vdd

rise time of less than 10V/s might trigger this problem.

The same problem might occur during a supply voltage drop during which Vdd remains between 300mV and 80mV for more than 200ms before going back to the specified Vdd level. As an example, having a residue battery voltage of less than 0.3V but more than 0.08V in a rechargeable battery application might trigger this problem when the charger providing the 3V supply is being connected.

Work-around: Apply another power-on Reset during which Vdd rises from below 0.8V to above 2.0V in less than 200ms.

Note.1: Increased power consumption from battery while RTC is running from the main 3.3V supply

Introduction: The RTC is powered by its own power supply pin, Vbat, which can be connected to a battery or to the same 3.3 volt supply used by the rest of the device.

Problem: If the VBAT is connected to an external battery, RTC will consume more power from the battery if the core is running and the selected clock sourceis the prescaler.

Work-around: Switch the clock source such that the RTC takes the clock from the 32 KHz oscillator that is connected to the RTCX1 and RTCX2 pins. After initialization of the RTC, clear the PCRTC bit in the PCONP register to switch off the peripheral clock (pclk) to RTC. Any further writes to the RTC would require this bit to be set. This will reduce the power consumed from VBAT.

Note.2: Port pin P0.31 must not be driven low during reset. If low on reset the device behaviour is undetermined.

Electrical and Timing Specification Deviations of the LPC2131

ESD.1: Philips Quality Spec specifies ESD-HBM should be above 2.0 kV. The LPC2131 passed ESD-HBM Stress up to 2.5kV but without VDD-to-VDD zapping (e.g. Vref to VBat, or Vref to V3A). Units zapped according to JEDEC Standard (JESDA22-A114-B) did not pass the ESD Post Stress Test.

From Revision C onwards, this issue has been fixed and the ESD-HBM limit has been improved. The LPC2131 ESD-HBM is now 4.0kV.