ES_LPC1850/30/20/10

Errata sheet LPC1850, LPC1830, LPC1820, LPC1810

Rev. 6.1 — 22 November 2013 **Errata sheet**

Document information

Info	Content
Keywords	LPC1850FET256; LPC1850FET180; LPC1830FET256; LPC1830FET180; LPC1830FET100; LPC1830FBD144; LPC1820FET100; LPC1820FBD144; LPC1810FET100; LPC1810FBD144; Cortex-M4 flashless, Rev A, C errata
Abstract	This errata sheet describes both the known 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.

Revision history

Rev	Date	Description
6.1	20131122	• Corrected retained memory location for SRAM.1 to 0x10088000.
6	20131021	• Added USBROM.2, USBROM.3, SRAM.1.
5	20130715	• Added IRC.1, IDDA. • Document name changed from ES_LPC18X0_A to ES_LPC18X0. • Added USB.1. • Added ISP.2. • Renamed CDC.1 to USBROM.1.
4	20130125	• Added I2C.1.
3.1	20121130	• Corrected part number typo in Section 3.7. • Added CDC.1. • Updated workaround for IBAT.1.
3	20121015	• Updated C_CAN.1. • Added Rev. C. • Removed AES.1, ETM.1, RGU.1, SPIFI.2; documented in user manual.
2.2	20120808	• Added IBAT.2 and RGU.1. • Corrected C_CAN0/C_CAN1 peripheral assignment.
2.1	20120713	• Added C_CAN.1.
2	20120601	• Added ISP.1, ETM.1, IAP.1, PMC.1 and IBAT.1.
1.3	20120401	• Updated SPIFI.1. • Added SPIFI.2. • Removed ADC.1 and USB0.1.
1.2	20120201	• Added OTP.1.
1.1	20120120	• Added ADC.1.
1	20111111	• Initial version.

Contact information

For more information, please visit: **http://www.nxp.com**

For sales office addresses, please send an email to: **salesaddresses@nxp.com**

1. Product identification

The LPC1850/30/20/10 devices (hereafter referred to as ‘LPC18x0’) typically have the following top-side marking:

LPC18x0xxxxxx

xxxxxxxx

xxxYYWWxR[x]

The last/second to last letter in the last line (field ‘R’) will identify the device revision. This Errata Sheet covers the following revisions of the LPC18x0:

Table 1. Device revision table

Revision identifier (R)	Revision description
‘A’	Initial device revision
‘C’	Second device revision

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

2. Errata overview

Table 2. Functional problems table

Functional problems	Short description	Revision identifier	Detailed description
BOOT.1	USB1 boot is not functional	‘A’	Section 3.1
C_CAN.1	Writes to CAN registers write through to other peripherals	‘A’, ‘C’	Section 3.2
I2C.1	In the slave-transmitter mode, the device set in the monitor mode must write a dummy value of 0xFF into the DAT register.	‘A’, ‘C’	Section 3.3
IAP.1	In-Application Programming API not present on flashless parts	‘A’, ‘C’	Section 3.4
IDDA.1	Increased IDDA current in low power modes	‘C’	Section 3.5
IRC.1	Increased IRC frequency variation	‘C’	Section 3.6
ISP.1	Part ID format incorrect	‘A’, ‘C’	Section 3.7
ISP.2	‘J’ command in ISP mode swaps last two items	‘C’ (with a boot ROM version of 11.2)	Section 3.8
MCPWM.1	MCPWM abort pin not functional	‘A’, ‘C’	Section 3.9
OTP.1	OTP ROM driver may not program boot source	‘A’, ‘C’	Section 3.10
PMC.1	PMC.x power management controller fails to wake up from deep sleep, power down, or deep power down	‘A’, ‘C’	Section 3.11
SPIFI.1	Rebooting from some SPIFI devices will be delayed by 60 seconds	‘A’	Section 3.12
SRAM.1	Misconfigured parts	‘A’, ‘C’ (with a date code ≤ 1315)	Section 3.13
USB.1	USB0 unable to communicate with low-speed USB peripheral in host mode when using full-speed hub	‘A’, ‘C’	Section 3.14

Table 2. Functional problems table *…continued*

Functional problems	Short description	Revision identifier	Detailed description
USBROM.1	The CDC class USB ROM drivers return a STALL condition	‘A’	Section 3.15
USBROM.2	Nested NAK handling of EP0 OUT endpoint	‘A’, ‘C’	Section 3.16
USBROM.3	Isochronous transfers	‘A’, ‘C’	Section 3.17

Table 3. AC/DC deviations table

AC/DC deviations	Short description	Product version(s)	Detailed description
IBAT.1	VBAT supply current higher than expected	‘A’, ‘C’	Section 4.1
IBAT.2	VBAT supply current higher than expected	‘A’	Section 4.2
PWR.1	Deep sleep and Power-down mode consume more current than expected	‘A’	Section 4.3
PWR.2	VDDIO consumes more current than expected	‘A’	Section 4.4

Table 4. Errata notes table

Errata notes	Short description	Revision identifier	Detailed description
n/a	n/a	n/a	n/a

3. Functional problems detail

3.1 BOOT.1

Introduction:

The internal ROM memory is used to store the boot code of the LPC18x0. After a reset, the ARM processor will start its code execution from this memory. The boot ROM memory includes the following features:

- …Boot from USB1….

Problem:

Boot from USB1 is not functional. This does not affect use of USB1 after bootup.

Work-around:

USB0 can be used to boot the part.

3.2 C_CAN.1

Introduction:

Controller Area Network (CAN) is the definition of a high performance communication protocol for serial data communication. The C_CAN controller is designed to provide a full implementation of the CAN protocol according to the CAN Specification Version 2.0B. The C_CAN controller allows to build powerful local networks with low-cost multiplex wiring by supporting distributed real-time control with a very high level of security.

Problem:

On the LPC18x0, there is an issue with the C_CAN controller AHB bus address decoding that applies to both C_CAN controllers. It affects the C_CAN controllers when peripherals on the same bus are used. Writes to the ADC, DAC, I2C, and I2S peripherals can update registers in the C_CAN controller. Specifically, writes to I2C0, MCPWM, and I2S can affect C_CAN1. Writes to I2C1, DAC, ADC0, and ADC1 can affect C_CAN0. The spurious C_CAN controller writes will occur at the address offset written to the other peripherals on the same bus. For example, a write to ADC0 CR register which is at offset 0 in the ADC, will result in the same value being written to the C_CAN0 CNTL register which is at offset 0 in the C_CAN controller. Writes to the C_CAN controller will not affect other peripherals.

Work-around:

Workarounds include: Using a different C_CAN peripheral. Peripherals I2C1, DAC, ADC0, and ADC1 can be used at the same time as C_CAN1 is active without any interference. The I2C0, MCPWM, and I2S peripherals can be used at the same time as C_CAN0 is active without any interference. Another workaround is to gate the register clock to the CAN peripheral in the CCU. This will prevent any writes to other peripherals from taking effect in the CAN peripheral. However, gating the CAN clock will prevent the CAN peripheral from operating and transmitting or receiving messages. This workaround is most useful if your application is modal and can switch between different modes such as an I2S mode and a CAN mode. Another workaround is to avoid writes to the peripherals while CAN is active. For example, the ADC could be configured to sample continuously or when triggered by a timer, before the CAN is configured. Afterwards, C_CAN0 can be used since the ADC will operate without requiring additional writes.

3.3 I2C.1

Introduction:

The I2C monitor allows the device to monitor the I2C traffic on the I^2C-bus in a non-intrusive way.

Problem:

In the slave-transmitter mode, the device set in the monitor mode must write a dummy value of 0xFF into the DAT register. If this is not done, the received data from the slave device will be corrupted. To allow the monitor mode to have sufficient time to process the data on the I2C bus, the device may need to have the ability to stretch the I2C clock. Under this condition, the I2C monitor mode is not 100% non-intrusive.

Work-around:

When setting the device in monitor mode, enable the ENA_SCL bit in the MMCTRL register to allow clock stretching.

Software code example to enable the ENA_SCL bit:

```
LPC_I2C_MMCTRL |= (1<<1); //Enable ENA_SCL bit
```

In the I2C ISR routine, for the status code related to the slave-transmitter mode, write the value of 0xFF into the DAT register to prevent data corruption. In order to avoid stretching the SCL clock, the data byte can be saved in a buffer and processed in the Main loop. This ensures the SI flag is cleared as fast as possible.

Software code example for the slave-transmitter mode:

```
case 0xA8:     // Own SLA + R has been received, ACK returned
case 0xB0:
case 0xB8:     // data byte in DAT transmitted, ACK received
case 0xC0:     // (last) data byte transmitted, NACK received
case 0xC8:     // last data byte in DAT transmitted, ACK received
    DataByte = LPC_I2C->DATA_BUFFER;//Save data. Data can be process in Main loop
    LPC_I2C->DAT = 0xFF;            // Pretend to shift out 0xFF
    LPC_I2C->CONCLR = 0x08;         // clear flag SI
break;
```

3.4 IAP.1

Introduction:

The LPC18x0 microcontrollers contain an API for In-Application Programming of flash memory. This API also allows identification of the part.

Problem:

On the LPC18x0 microcontrollers, the IAP API is not present. The ISP interface is present which allows the part to be identified externally (via the UART) but part identification is not possible internally using the IAP call because it is not implemented.

Work-around:

To detect whether or not the IAP API is present, check the IAP entry point at 0x10400100. If it is set to 0x12345678, the part does not implement IAP. The first word of the Part ID can be read directly from OTP at 0x40045000. The second word of the Part ID is always '0' on flashless parts.

3.5 IDDA.1

Introduction:

This part includes two 10-bit analog to digital converters that require a 3.3 V supply.

Problem:

While in deep sleep and power-down modes, analog domains show increased current. The table below shows the actual power drawn in these modes.

Work-around:

None.

Table 5. Static characteristics

T_{amb} = −40 °C to +85 °C, unless otherwise specified

Symbol	Parameter	Conditions		Min	Typ	Max	Unit
I_{DDA}	analog supply current	deep sleep mode	[1]	-	4.2	-	µA
		power-down mode	[1]	-	4.2	-	µA
		deep power-down mode	[1]	-	0.007	-	µA

[1] $V_{DDA(3V3)}$ = 3.3 V; T_{amb} = 25 °C.

3.6 IRC.1

Introduction:

The IRC is used as the clock source for the WWDT and can also be used as PLL input The nominal IRC frequency is 12 MHz. The IRC is trimmed to 1 % accuracy over the entire voltage and temperature range.

Problem:

On LPC18x0 flashless devices, the IRC deviates at high temperatures. This results in worse IRC accuracy than specified in the data sheet.

Work-around:

When a higher timing accuracy is required use of an external crystal is recommended.

Table 6. Dynamic characteristic: IRC oscillator

2.2 V ≤ $V_{DD(REG)(3V3)}$ ≤ 3.6 V.

Symbol	Parameter	Conditions	Min	Typ[1]	Max	Unit
$f_{osc(RC)}$	internal RC oscillator frequency	−40 °C ≤ T_{amb} < 0 °C	11.76	12.0	12.24	MHz
		0 °C ≤ T_{amb} ≤ 85 °C	11.88	12.0	12.12	MHz

[1] Typical ratings are not guaranteed. The values listed are at room temperature (25 °C), nominal supply voltages.

3.7 ISP.1

Introduction:

A reduced set of In-System-Programming (ISP) commands are supported for flashless parts. The ISP 'J' command can be used to query the part identification number.

Problem:

On the LPC18x0 microcontrollers, the J command returns incorrectly formatted data. Instead of returning two words (plus the return code) as specified in the User's Manual, IAP command 54 and ISP command 'J' only return a single word (plus return code). That single word contains the first word of the part identification number with the first 16 bits swapped with the last 16 bits. For example, an LPC1850FET256 will return 0x0830A000 instead of the correct value, 0xA0000830.

Work-around:

When using ISP, if only one word of data is returned, swap the two 16-bit segments of the word and assume the second word of data is 0.

3.8 ISP.2

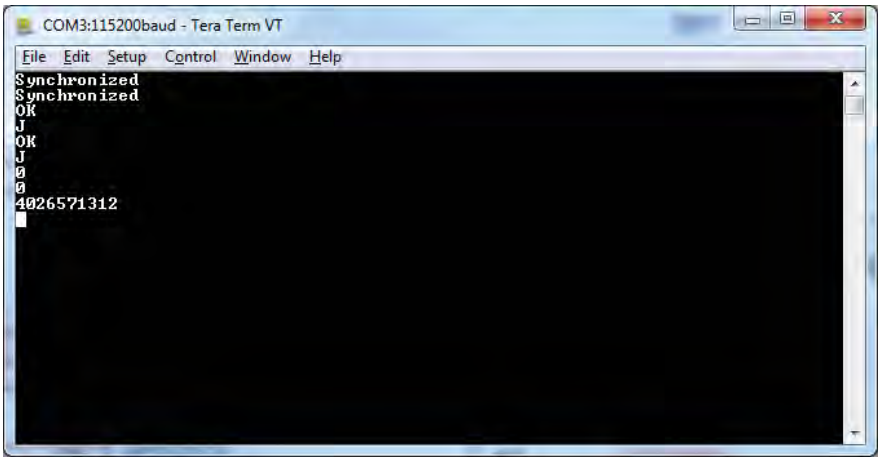
Introduction:

All LPC18x0 parts include a feature called In-System Programming (ISP) which boots up over the UART port and provides a terminal-based communication mechanism to query certain characteristics of the part. One of these is the ability to retrieve the Part Identification number.

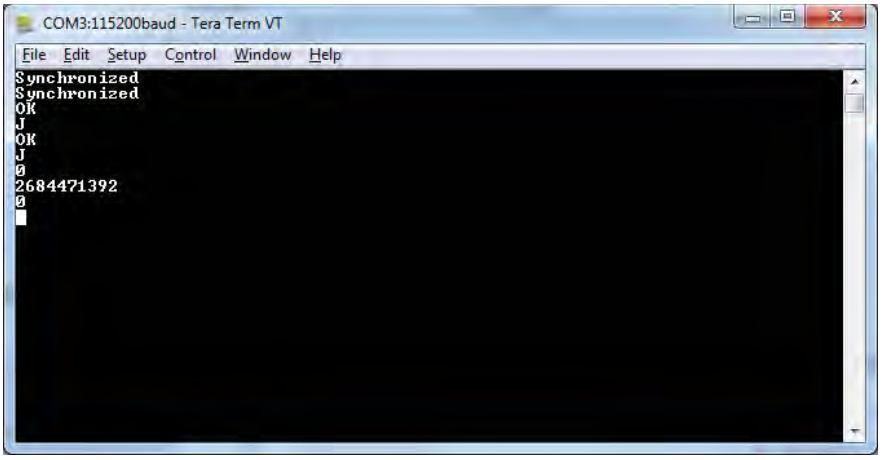
Problem:

The 'J' command in ISP mode should return an error code, followed by an ASCII string representation of the part ID, followed by a 0. However what is actually returned is the error code, followed by a 0, followed by an ASCII string representation of the part ID. The problem is the last two items returned are swapped.

Incorrect example:



Correct example:



Work-around:

There is no work-around for this problem.

3.9 MCPWM.1

Introduction:

The Motor Control PWM engine is optimized for three-phase AC and DC motor control applications, but can be used in many other applications that need timing, counting, capture, and comparison. The MCPWM contains a global Abort input that can force all of the channels into a passive state and cause an interrupt.

Problem:

The MCPWM Abort input is not functional.

Work-around:

The MCPWM Abort function can be emulated in software with the use of a non-maskable interrupt combined with an interrupt handler that shuts down the PWM. This will result in a small delay on the order of 50 main clock cycles or about 1/3 of a microsecond at 150 MHz. Alternatively, the State Configurable Timer (SCT) can be configured to implement MCPWM functionality including an Abort input. The SCT can respond to external inputs in one clock cycle.

3.10 OTP.1

Introduction:

The LPC18x0 contain OTP memory which can configure the boot source, as well as a set of routines in ROM to program the boot source into OTP memory.

Problem:

There is a problem in the OTP boot source programming code in ROM which requires registers to be initialized in order to ensure successful boot source OTP programming.

Work-around:

1. Add this function to your program.

```
void OTP_fix(volatile unsigned dummy0,volatile unsigned dummy1,volatile unsigned
dummy2,volatile unsigned dummy3)
{
}
```

2. Call this function before calling otp_ProgBootSrc.

```
rval = otp_Init();
OTP_fix(0,0,0,0);
rval = otp_ProgBootSrc(OTP_BOOTSRC_SPIFI);
```

This will be fixed in the next boot ROM revision.

3.11 PMC.1

Introduction:

The PMC implements the control sequences to enable transitioning between different power modes and controls the power state of each peripheral. In addition, wake-up from any of the power-down modes based on hardware events is supported.

Problem:

When the chip is in a transition from active to Deep Sleep, Power Down, or Deep Power Down, wakeup events are not captured and they will block further wakeup events from propagating. The time window for this transition is 6 uS and is not affected by the chip clock speed. After a wakeup event is received during the PMC transition, the chip can only recover by using an external hardware reset or by cycling power.

Work-around:

Make sure that a wakeup signal is not received during the Deep Sleep, Power Down, or Deep Power Down transition period. An example circuit to work around this could include an external 6 μS one shot which could be triggered via software using a GPIO line when entering Deep Sleep, Power Down, or Deep Power Down mode. The one-shot's output could be used to gate the wakeup signal(s) to prevent receiving a wakeup signal during the PMC transition period. Depending on the system design, it may also be needed to latch the wakeup signal(s) so that they will still be present after the one-shot's 6 μS timeout.

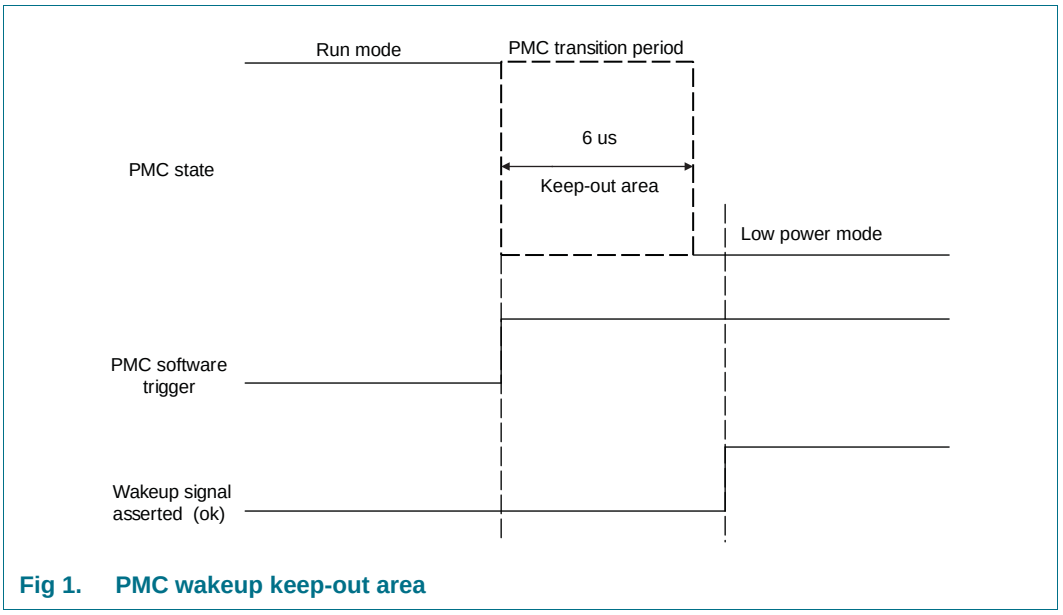


Fig 1. PMC wakeup keep-out area

3.12 SPIFI.1

Introduction:

The boot ROM includes a SPI Flash Interface (SPIFI) driver that allows low-cost serial flash memories to be connected to the ARM Cortex-M3 processor with little performance penalty compared to parallel flash devices with higher pin count. SPIFI provides a memory-mapped area where the contents of the external serial flash memory appear.

Problem:

The boot ROM does not properly re-initialize the external serial flash device if it is already set up for "no opcode" or "continuous read" mode. This affects use after unplanned resets such as a hardware reset or watchdog timer reset. Booting from SPIFI is affected and may not be successful until after the 60 second boot failure timeout if the external serial flash device is in "no opcode" or "continuous read" mode.

Work-around:

During a planned reboot, remove the external QSPI flash from no opcode mode before resetting the CPU by using the SPIFI driver library's cancel_mem_mode call. The SPIFI driver library is available from lpcware.com. In the event of an unplanned reset, the driver will initialize the flash device if it is called a second time so an external watchdog could be provided to reset the CPU in case of boot failure from SPIFI. Finally there is a built-in 60-second boot timeout which will result in a successful boot after one minute in the event of a failure.

3.13 SRAM.1

Introduction:

The LPC18x0 parts all have an 8 kB region of internal SRAM that is retained during Sleep, Deep-sleep, and Power-down modes. This retained memory is located at 0x10088000.

Problem:

Parts with date codes marked ≤ 1315 locate the 8 kB of retained memory at 0x10090000. Also, the part numbers for these parts returned by the ISP Read Part Identification command is incorrect.

The correct part numbers look like this:

Device	Hex coding	
	Word0	Word1
LPC1850FET256	0xF000 D830	0x0000 0000
LPC1850FET180	0xF000 D830	0x0000 0000
LPC1850FBD208	0xF000 D830	0x0000 0000
LPC1830FET256	0xF000 DA30	0x0000 0000
LPC1830FET180	0xF000 DA30	0x0000 0000
LPC1830FET100	0xF000 DA30	0x0000 0000
LPC1830FBD144	0xF000 DA30	0x0000 0000
LPC1820FET100	0xF00A DB3C	0x0000 0000
LPC1820FBD144	0xF00A DB3C	0x0000 0000
LPC1820FBD100	0xF00A DB3C	0x0000 0000
LPC1810FET100	0xF00B 5B3F	0x0000 0000
LPC1810FBD144	0xF00B 5B3F	0x0000 0000
LPC4320FET100	0xA000 CB3C	0x0000 0000
LPC4320FBD144	0xA000 CB3C	0x0000 0000
LPC4310FET100	0xA00A CB3F	0x0000 0000
LPC4310FBD144	0xA00A CB3F	0x0000 0000

Fig 2. Correct part numbers

The misconfigured parts discussed in this errata have part numbers that look like this:

Device	Hex coding	
	Word0	Word1
LPC1850FET256	0xF000 9830	0x0000 0000
LPC1850FET180	0xF000 9830	0x0000 0000
LPC1850FBD208	0xF000 9830	0x0000 0000
LPC1830FET256	0xF000 9A30	0x0000 0000
LPC1830FET180	0xF000 9A30	0x0000 0000
LPC1830FET100	0xF000 9A30	0x0000 0000
LPC1830FBD144	0xF000 9A30	0x0000 0000
LPC1820FET100	0xF00A 9B3C	0x0000 0000
LPC1820FBD144	0xF00A 9B3C	0x0000 0000
LPC1820FBD100	0xF00A 9B3C	0x0000 0000
LPC1810FET100	0xF00B 1B3F	0x0000 0000
LPC1810FBD144	0xF00B 1B3F	0x0000 0000
LPC4320FET100	0xA000 8B3C	0x0000 0000
LPC4320FBD144	0xA000 8B3C	0x0000 0000
LPC4320FBD100	0xA000 8B3C	0x0000 0000
LPC4310FET100	0xA00A 8B3F	0x0000 0000
LPC4310FBD144	0xA00A 8B3F	0x0000 0000

Fig 3. Misconfigured part numbers

Work-around:

There is no way to change the memory configuration of these parts. However it is advised that software check the value of the part number returned from the ISP Read Part Identification command and take the appropriate action based on what value is read. If bit 14 equals 0 then the part is misconfigured as described in this errata.

3.14 USB.1

Introduction:

The LPC18x0 parts include two USB 2.0 controllers that can operate in host mode at high-speed. One of these controllers, USB0, contains an on-chip high-speed UTMI+ compliant transceiver (PHY) which supports high-speed, full-speed, and low-speed USB-compliant peripherals.

Problem:

The USB controller called USB0 is unable to communicate with a low-speed USB peripheral in host mode when there is a full-speed hub directly connected to the USB0 port and a low-speed peripheral is connected in the tree somewhere below this full-speed hub. Only USB0 has this problem; the other USB controller, USB1 does not.

Work-around:

There is no work-around for this problem. It is suggested that the low-speed USB peripheral is either connected directly to USB0 or a high-speed hub is placed between that peripheral and USB0.

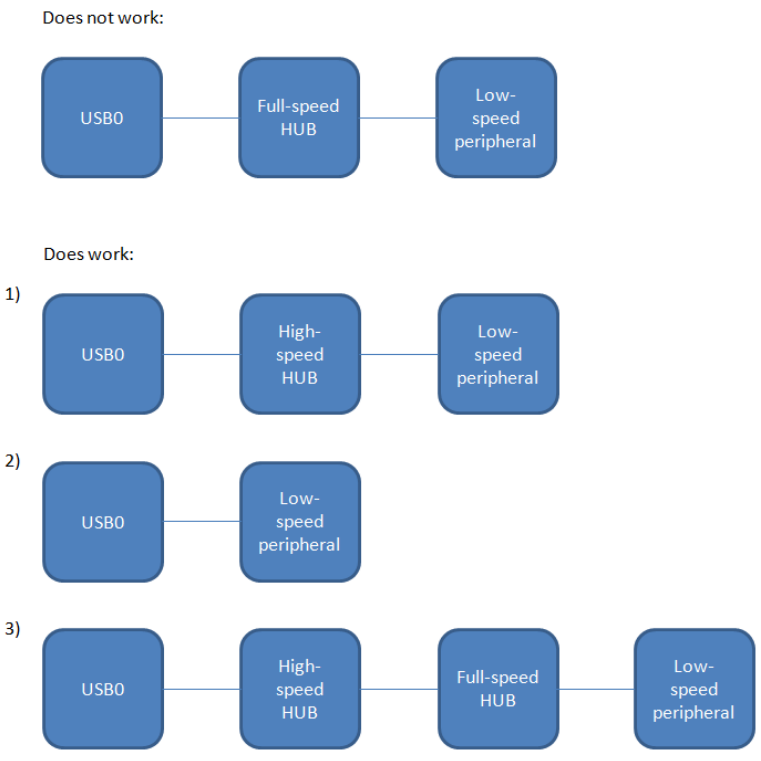


Fig 4. Suggested USB0 to low-speed peripheral connections

3.15 USBROM.1

Introduction:

The CDC class USB ROM drivers have a bug that causes some Windows terminal emulation programs like Hyperterm to fail to connect. This bug only exists in revision 'A' parts.

Problem:

The CDC class USB ROM drivers return a STALL condition after the DATA stage of a Set Line Coding CONTROL transfer from the host. The correct behavior is to return an ACK.

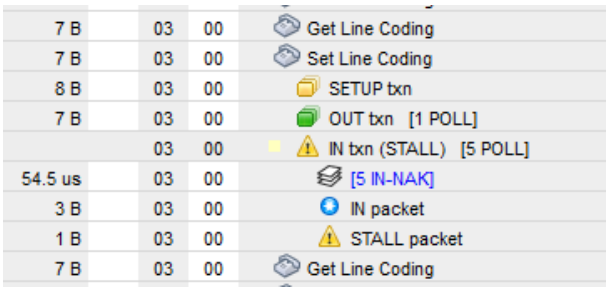


Fig 5. Incorrect

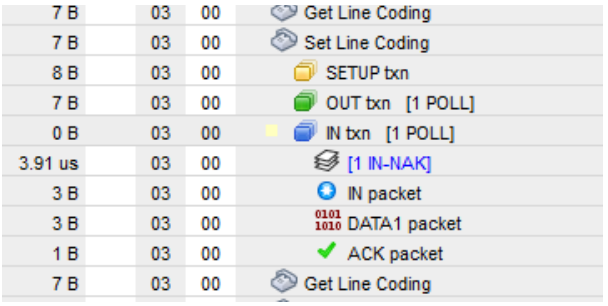


Fig 6. Correct

Work-around:

The way to work around this bug is to override the default CDC class specific endpoint 0 handler and handle the processing of OUT packets in your application. Use the following source code in red to accomplish this.

```
USB_EP_HANDLER_T g_defaultCdcHdlr; // default CDC handler
ErrorCode_t CDC_ep0_override_hdlr(USBD_HANDLE_T hUsb, void* data, uint32_t event)
{
  USB_CORE_CTRL_T* pCtrl = (USB_CORE_CTRL_T*)hUsb;
  USB_CDC_CTRL_T* pCdcCtrl = (USB_CDC_CTRL_T*) data;
  ErrorCode_t ret = ERR_USBD_UNHANDLED;

  if( (event == USB_EVT_OUT) &&
      (pCtrl->SetupPacket.bmRequestType.BM.Type == REQUEST_CLASS) &&
      (pCtrl->SetupPacket.bmRequestType.BM.Recipient == REQUEST_TO_INTERFACE) &&
      ((pCtrl->SetupPacket.wIndex.WB.L == pCdcCtrl->cif_num) || /* IF number correct?
     */
      (pCtrl->SetupPacket.wIndex.WB.L == pCdcCtrl->dif_num)) ) {

    pCtrl->EP0Data.pData -= pCtrl->SetupPacket.wLength;
    ret = pCdcCtrl->CIC_SetRequest(pCdcCtrl, &pCtrl->SetupPacket,
     &pCtrl->EP0Data.pData,
       pCtrl->SetupPacket.wLength);
    if ( ret == LPC_OK) {

      USBD_API->core->StatusInStage(pCtrl);                    /* send Acknowledge */
    }
```

```
  } else {
      ret = g_defaultCdcHdlr(hUsb, data, event);
  }
   return ret;
}
void UsbdCdc_Init(void)
{
      .
      .
      .

      USBD_API->cdc->init(UsbHandle, &cdc_param, &UsbdCdcHdlr);

      {
            // This code must be placed immediately after the call to
      USBD_API->cdc->init()
            USB_CORE_CTRL_T* pCtrl = (USB_CORE_CTRL_T*)UsbHandle;
            /* store the default CDC handler and replace it with ours */
            g_defaultCdcHdlr = pCtrl->ep0_hdlr_cb[pCtrl->num_ep0_hdlrs - 1];
            pCtrl->ep0_hdlr_cb[pCtrl->num_ep0_hdlrs - 1] = CDC_ep0_override_hdlr;
      }
}
```

3.16 USBROM.2

Introduction:

The USB ROM drivers include a default endpoint 0 handler which acts on events generated by the USB controller as a result of traffic occurring over the control endpoint. The user has the option of overloading this default handler for the purpose of performing user specific processing of control endpoint traffic as required.

One of the actions the default endpoint 0 handler performs is to prepare the DMA engine for data transfer after the controller has sent out a NAK packet to the host controller. This is done in preparation for the arrival of the next OUT request received from the host.

Problem:

Due to a race condition there is the chance that a second NAK event will occur before the default endpoint0 handler has completed its preparation of the DMA engine for the first NAK event. This can cause certain fields in the DMA descriptors to be in an invalid state when the USB controller reads them, thereby causing a hang.

Work-around:

Override the default endpoint 0 handler to add checks for and prevents nested NAK event processing activity.

This is an example of how to do this:

```
// Endpoint 0 patch that prevents nested NAK event processing
static uint32_t g_ep0RxBusy = 0; /* flag indicating whether EP0 OUT/RX buffer is
busy. */
static USB_EP_HANDLER_T g_Ep0BaseHdlr; /* variable to store the pointer to base EP0
handler */

/*-------------------------------------------------------------------------
  EP0_patch :
 *-------------------------------------------------------------------------*/
ErrorCode_t EP0_patch(USBD_HANDLE_T hUsb, void* data, uint32_t event)
{

    switch (event) {
        case USB_EVT_OUT_NAK:
            if (g_ep0RxBusy) {
                /* we already queued the buffer so ignore this NAK event. */
                return LPC_OK;
            } else {
                /* Mark EP0_RX buffer as busy and allow base handler to queue the
     buffer. */
                g_ep0RxBusy = 1;
            }
            break;
        case USB_EVT_SETUP: /* reset the flag when new setup sequence starts */
        case USB_EVT_OUT:
            /* we received the packet so clear the flag. */
            g_ep0RxBusy = 0;
```

```
            break;
    }
    return g_Ep0BaseHdlr(hUsb, data, event);
}


// Install the endpoint 0 patch immediately after USB initialization via the
     hw->Init() call.

*-------------------------------------------------------------------------
  usbd_init: usb subsystem init routine
*-------------------------------------------------------------------------*/
ErrorCode_t usbd_init (void)
{
    USBD_API_INIT_PARAM_T usb_param;
    USB_CORE_DESCS_T desc;
    ErrorCode_t ret = LPC_OK;
    USB_CORE_CTRL_T* pCtrl;


…
    /* USB Initialization */
    ret = USBD_API->hw->Init(&g_AdcCtrl.hUsb, &desc, &usb_param);
    if (ret == LPC_OK) {

        /* register EP0 patch */
        pCtrl= (USB_CORE_CTRL_T*)g_AdcCtrl.hUsb; /* convert the handle to control
     structure */
        g_Ep0BaseHdlr = pCtrl->ep_event_hdlr[0]; /* retrieve the default EP0_OUT
     handler */
        pCtrl->ep_event_hdlr[0] = EP0_patch;     /* set our patch routine as EP0_OUT
     handler */


….'
  }
….'
  return LPC_OK;
}
```

3.17 USBROM.3

Introduction:

The USB ROM drivers configure and manage data structures used by the USB controller's DMA engine to move data between the controller's internal fifos and system memory. The configuration of these data structures are based on many parameters including the type of transfer, control, bulk, interrupt, or isochronous, that is to be performed. These data structures reside in system RAM on a 2 kB boundary and are pointed to by the ENDPOINTLISTADDR register.

Problem:

The USB ROM drivers incorrectly configures the Endpoint Capabilities/Characteristics field of the device Queue Head (dQH) structure for isochronous endpoints. Specifically, the MULT member is set to 0 and the ZLT member is set to 1. Also if the maximum size of isochronous packets are 1024 bytes the Max_packet_length member will be set to 0. For any other packet size this member is set correctly.

Work-around:

To use isochronous transfers with the USB ROM drivers the Endpoint Capabilities/Characteristics field must be correctly configured for that endpoint's device Queue Head structure. The USB ROM driver always sets this field (incorrectly) when the host sends a Set Interface control packet and then it calls the USB_Interface_Event callback routine, so the field must be set with the proper value in this callback routine.

This is the device Queue Head structure:

```
typedef volatile struct
{
  volatile uint32_t cap;
  volatile uint32_t curr_dTD;
  volatile uint32_t next_dTD;
  volatile uint32_t total_bytes;
  volatile uint32_t buffer0;
  volatile uint32_t buffer1;
  volatile uint32_t buffer2;
  volatile uint32_t buffer3;
  volatile uint32_t buffer4;
  volatile uint32_t reserved;
  volatile uint32_t setup[2];
  volatile uint32_t gap[4];
}  DQH_T;
```

This is an Interface Event callback routine:

```
ErrorCode_t USB_Interface_Event (USBD_HANDLE_T hUsb)
{
     USB_CORE_CTRL_T* pCtrl = (USB_CORE_CTRL_T*)hUsb;
     uint16_t wIndex = pCtrl->SetupPacket.wIndex.W; // Interface number
     uint16_t wValue = pCtrl->SetupPacket.wValue.W; // Alternate setting number

     if (wIndex == isochronous_interface_number && wValue == 1)
     {
```

```
            DQH_T* ep_QH = *(DQH_T**)0x40006158; // ENDPOINTLISTADDR register
            int QH_idx = ((endpoint_address & 0x0F) << 1) + 1;

            ep_QH[QH_idx].cap = ((packets_executed_per_transaction_descriptor << 30) |
      (maximum_packet_size << 16));
      }

  return LPC_OK;
}
```

The value of isochronous_interface_number should correspond to the interface number in the USB descriptor that holds the isochronous endpoint you wish to use.

The value of maximum_packet_size should correspond to the wMaxPacketSize member of the isochronous endpoint descriptor

The value of endpoint_address should correspond to the bEndpointAddress member of the isochronous endpoint descriptor

4. AC/DC deviations detail

4.1 IBAT.1

Introduction:

The LPC18x0 contain a Real-Time Clock which measures the passage of time. The RTC has an ultra-low power design to support battery powered systems with a dedicated battery supply pin.

Problem:

On the LPC18x0, high current consumption of about 70 μA may occur on the VBAT power supply pin.

Work-around:

VBAT current consumption can be lowered significantly by configuring the RTC_ALARM pin as "Inactive" by setting the ALARMCTRL 7:6 field in CREG0 to 0x3. These bits persist through power cycles and reset while VBAT is present.

For CREG0[13:12] reserved value 0x3 should be used; this value should be set once after a power on reset.

4.2 IBAT.2

Introduction:

The LPC18x0 contain a Real-Time Clock which measures the passage of time. The RTC has an ultra-low power design to support battery powered systems with a dedicated battery supply pin.

Problem:

On the LPC18x0, high current consumption of about 15 μA may occur on the VBAT power supply pin despite applying the workaround in IBAT.1.

Work-around:

The problem is caused by a design error and there is currently no work-around.

4.3 PWR.1

Introduction:

The LPC18x0 contain several low-power modes. The PMC implements the control sequences to enable transitioning between different power modes and controls the power state of each peripheral.

Problem:

A design error results in about 15 μA higher current consumption during Deep Sleep and Power Down mode.

Work-around:

None.

4.4 PWR.2

Introduction:

The LPC18x0 contain several low-power modes. Most of the low power modes are designed to trigger external shut down of VDDIO using an external switch for lowest power consumption.

Problem:

A design error results in about 60 µA higher current consumption on VDDIO during all active and low power modes except for deep power down.

Work-around:

VDDIO can be switched off upon entry to low power modes with addition of an external switch.

5. Legal information

5.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

5.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

5.3 Trademarks

Notice: All referenced brands, product names, service names and trademarks are the property of their respective owners.

6. Contents

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 22 November 2013
Document identifier: ES_LPC18X0