ES_LPC15xx

Errata sheet LPC15xx

Rev. 2.4 — 7 March 2018 Errata sheet

Document information

Info	Content
Keywords	LPC1549JBD100; LPC1549JBD64; LPC1549JBD48; LPC1548JBD100; LPC1548JBD64; LPC1547JBD64; LPC1547JBD48; LPC1519JBD100; LPC1519JBD64; LPC1518JBD100; LPC1518JBD64; LPC1517JBD64; LPC1517JBD48
Abstract	LPC15xx errata

Revision history

Rev	Date	Description
2.4	20180307	• Added USB ROM.2
2.3	20170807	• Added USB ROM.1
2.2	20170217	• Added RTC.1
2.1	20151120	• Added DPD.1
2	20150417	• Added UART.1
1	20140224	• 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 LPC15xx devices typically have the following top-side marking:

LPC15xxxxxxx

xxxxxxx

xxxYYWWxR

The last letter in the last line (field ‘R’) will identify the device revision. This Errata Sheet covers the following revisions of the LPC15xx:

Table 1. Device revision table

Revision identifier (R)	Revision description
‘A’	Initial 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
USB.1	USB controller is unable to generate STALL on EP0_OUT.	‘A’	Section 3.1
I2C_ROM.1	Slave transmit ROM API functions not working.	‘A’	Section 3.2
UART.1	The UART controller sets the Idle status bits for receive and transmit before the transmission of the stop bit is complete.	‘A’	Section 3.3
DPD.1	Deep power-down mode is not functional outside certain voltage and temperature ranges.	‘A’	Section 3.4
RTC.1	The Real Time Clock (RTC) does not work reliably.	‘A’	Section 3.5
USB_ROM.1	FRAME_INT is cleared if new SetConfiguration or USB_RESET are received.	‘A’	Section 3.6
USB_ROM.2	USB full-speed device fail in the Command/Data/Status Flow after bus reset and bus re-enumeration.	‘A’	Section 3.7

Table 3. AC/DC deviations table

AC/DC deviations	Short description	Revision identifier	Detailed description
n/a	n/a	n/a	n/a

Table 4. Errata notes

Note	Short description	Revision identifier	Detailed description
n/a	n/a	n/a	n/a

3. Functional problems detail

3.1 USB.1: USB controller is unable to generate STALL on EP0_OUT

Introduction:

The LPC15xx have a full-speed USB device controller with support for 10 physical endpoints.

Problem:

The USB device controller is unable to return a STALL handshake on an OUT data packet to endpoint zero. An NAK handshake is returned instead.

Work-around:

Endpoint zero is the control endpoint. All requests sent to the control endpoint consist of three stages (SETUP / DATA / STATUS). When an unsupported ControlWrite request (with data phase) is sent by the host to the device, the device is unable to STALL the data phase of this request.

To solve this problem, the device firmware must accept the data transmitted during the data phase of this ControlWrite request and return a STALL handshake when the IN token for the STATUS stage is received.

3.2 I2C_ROM.1: Slave transmit ROM API functions not working

Introduction:

The LPC15XX provide a ROM API for transmitting data via the I^{2}C-bus interface in slave mode using the interrupt or polling method.

Problem:

The following I2C ROM API functions transmit 0xFF instead of the provided data:

- `ErrorCode_t i2c_slave_transmit_intr( I2C_HANDLE_T* h_i2c, I2C_PARAM* ptp, I2C_RESULT* ptr)`
- `ErrorCode_t i2c_slave_transmit_poll( I2C_HANDLE_T*  h_i2c, I2C_PARAM*  ptp, I2C_RESULT* ptr)`

Work-around:

Do not use the I2C slave transmit ROM API functions. The I2C register interface allows direct programming of the data transmit operation in slave mode.

3.3 UART.1

Introduction:

In receive mode, the UART controller provides a status bit (the RXIDLE bit in the UART STAT register) to check whether the receiver is currently receiving data. If RXIDLE is set, the receiver indicates it is idle and does not receive data.

In transmit mode, the UART controller provides two status bits (TXIDLE and TXDISSTAT bits in the UART STAT register) to indicate whether the transmitter is currently transmitting data. The TXIDLE bit is set by the controller after the last stop bit has been transmitted. The TXDISSTAT bit is set by the controller after the transmitter has sent the last stop bit and has become fully idle following a transmit disable executed by setting the TXDIS bit in the UART CTRL register.

The status bits can be used to implement software flow control, but their setting does not affect normal UART operation.

Problem:

The RXIDLE bit is incorrectly set for a fraction of the clock cycle between the reception of the last data bit and the reception of the start bit of the next word, that is while the stop bit is received. RXIDLE is cleared at the beginning of the start bit.

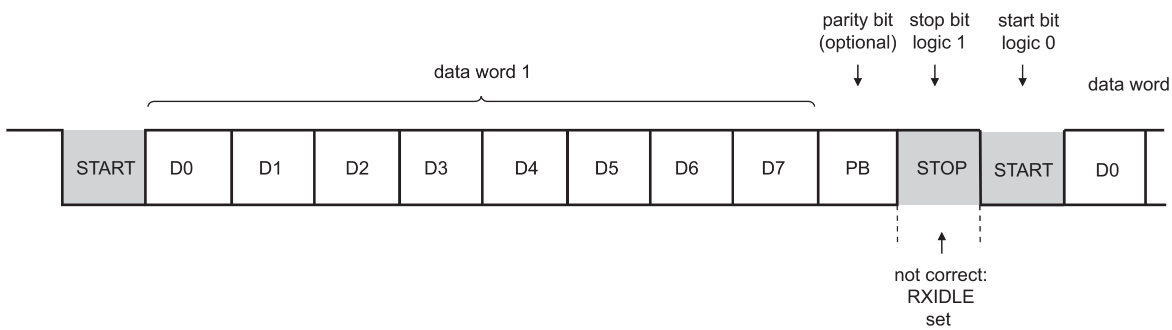

Fig 1. Incorrect setting of RXIDLE during UART receive

Both, TXIDLE and TXDISSTAT are set incorrectly between the last data bit and the stop bit while the transfer is still ongoing.

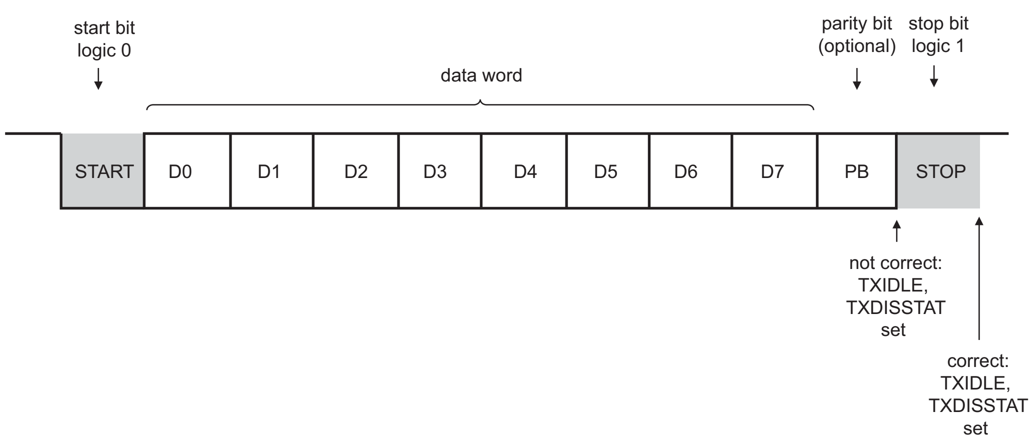

Fig 2. Incorrect setting of TXIDLE and TXDISSTAT during UART transmit

Work-around:

When writing code that checks for the setting of any of the status bits RXIDLE, TXIDLE, TXDISSTAT, check the value of the status bit in the STAT register:

- If status bit = 1, add a delay of one UART bit time (if STOPLEN = 0, one stop bit) or two bit times (if STOPLEN = 1, two stop bits) and check the value of the status bit again:
  - If status bit = 1, the receiver is idle.
  - If status bit = 0, the receiver is receiving data.
  - If the status bit = 0, the receiver is receiving data.

3.4 DPD.1

Introduction:

The LPC15xx has a supply voltage (V_{DD}) from 2.4 V to 3.6 V and can operate from -40 °C to 105 °C. The LPC15xx supports four reduced power modes (sleep, deep-sleep, power-down, and deep power-down mode). Deep power-down mode allows for maximal power savings where the entire system is shut down except for the general purpose registers in the PMU and the self wake-up timer. Only the general purpose registers in the PMU maintain their internal states in deep power-down mode.

Problem:

At temperatures ≤ 25 °C, the deep power-down mode is not functional if the V_{DD} supply voltage is > 3.4 V. At temperatures > 25 °C, the deep power-down mode is not functional if the V_{DD} supply voltage is > 3.35 V.

Work-around:

Deep power-down mode operates correctly for the entire temperature range (-40 °C to 105 °C) if the V_{DD} supply is between 2.4 V and 3.35 V. For temperatures ≤ 25 °C, ensure that the supply voltage is not > 3.4 V (V_{DD} = 2.4 V to 3.4 V) when using deep power-down mode. For temperatures > 25 °C, ensure that the supply voltage is not > 3.35 V (V_{DD} = 2.4 V to 3.35 V) when using deep power-down mode.

3.5 RTC.1: Real Time Clock (RTC) does not work reliably.

Introduction:

The RTC is a set of counters for measuring time when system power is on, and optionally when it is off. The RTC is clocked by a separate 32 kHz oscillator that produces a 1 Hz internal time reference.

Problem:

The RTC COUNT should increment by one in one second. With the problem, the RTC COUNT might increment by more than one within one second, causing inaccurate time counting.

Work-around:

None.

3.6 USB_ROM.1: FRAME_INT is cleared if new SetConfiguration or USB_RESET are received.

Introduction:

In the USB ROM API, the function call EnableEvent can be used to enable and disable FRAME_INT.

Problem:

When the FRAME_INT is enabled through the USB ROM API call:

```
ErrorCode_t(* USBD_HW_API::EnableEvent)(USBD_HANDLE_T hUsb, uint32_t EPNum, uint32_t
     event_type, uint32_t enable),
```

the FRAME_INT is cleared if new SetConfiguration or USB_RESET are received.

Work-around:

Implement the following software work-around in the ISR to ensure that the FRAME_INT is enabled:

```
void USB_IRQHandler(void)
{
USBD_API->hw->EnableEvent(g_hUsb, 0, USB_EVT_SOF, 1);
USBD_API->hw->ISR(g_hUsb);
}
```

3.7 USB_ROM.2: USB full-speed device fail in the Command/Data/Status Flow after bus reset and bus re-enumeration

Introduction:

The LPC15xx device family includes a USB full-speed interface that can operate in device mode and also, includes USB ROM based drivers. A Bulk-Only Protocol transaction begins with the host sending a CBW to the device and attempting to make the appropriate data transfer (In, Out or none). The device receives the CBW, checks and interprets it, attempts to satisfy the request of the host, and returns status via a CSW.

Problem:

When the device fails in the Command/Data/Status Flow, and the host does a bus reset / bus re-enumeration without issuing a Bulk-Only Mass Storage Reset, the USB ROM driver does not re-initialize the MSC variables. This causes the device to fail in the Command/Data/Status Flow after the bus reset / bus re-enumeration.

Work-around:

Implement the following software work-around to re-initialize the MSC variables in the USBD stack.

```
void  *g_pMscCtrl;

ErrorCode_t mwMSC_Reset_workaround(USBD_HANDLE_T hUsb)

{

((USB_MSC_CTRL_T *)g_pMscCtrl)->CSW.dSignature = 0;

     ((USB_MSC_CTRL_T *)g_pMscCtrl)->BulkStage = 0;

     return LPC_OK;

}

ErrorCode_t mscDisk_init(USBD_HANDLE_T hUsb, USB_CORE_DESCS_T *pDesc,
     USBD_API_INIT_PARAM_T *pUsbParam)

{    USBD_MSC_INIT_PARAM_T msc_param;

     ErrorCode_t ret = LPC_OK;

     memset((void *) &msc_param, 0, sizeof(USBD_MSC_INIT_PARAM_T));

     msc_param.mem_base = pUsbParam->mem_base;

     msc_param.mem_size = pUsbParam->mem_size;

     g_pMscCtrl = (void *)msc_param.mem_base;

     ret = USBD_API->msc->init(hUsb, &msc_param);

     /* update memory variables */

     pUsbParam->mem_base = msc_param.mem_base;

     pUsbParam->mem_size = msc_param.mem_size;
```

```
        return ret;
    }
        usb_param.USB_Reset_Event = mwMSC_Reset_workaround;
        ret = USBD_API->hw->Init(&g_hUsb, &desc, &usb_param);
```

4. AC/DC deviations detail

No known errata.

5. Errata notes

No known errata.

6. Legal information

6.1 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.

6.2 Disclaimers

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.

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.

6.3 Trademarks

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

7. Contents

Date of release: 7 March 2018
Document identifier: ES_LPC15XX