LPC546xx

Errata sheet LPC546xx

Rev. 2.4 — 25 January 2024 **Errata**

Document information

Information	Content
Keywords	LPC54605J256BD100, LPC54605J512BD100, LPC54605J256ET100, LPC54605J512ET100, LPC54606J256ET100, LPC54606J256BD100, LPC54606J512ET100, LPC54606J512BD100, LPC54616J512ET100, LPC54616J512BD100, LPC54605J256ET180, LPC54605J512ET180, LPC54606256ET180, LPC54606J512BD208, LPC54607J256ET180, LPC54607J512ET180, LPC54607J256BD208, LPC54608J512ET180, LPC54608J512BD208, LPC54616J256ET180, LPC54616J512BD208, LPC54618J512ET180, LPC54618J512BD208, LPC54628J512ET180
Abstract	LPC546xx errata

1 Product identification

The LPC546xx TFBGA180 and TFBGA100 packages have the following top-side marking:

- First line: ES_LPC546xxJyyy
  - yyy: flash size
- Second line: ET180 or ET100
- Third line: xxxxxxxxxxxx
- Fourth line: xxxyywwx[R]x
  - yyww: Date code with yy = year and ww = week.
  - xR = boot code version and device revision.

The LPC546xx LQFP208 and LQFP100 packages have the following top-side marking:

- First line: ES_LPC546xxJyyy
  - yyy: flash size
- Second line: BD208 or BD100
- Third line: xxxxxxxxxxxx
- Fourth line: xxxyywwx[R]x
  - yyww: Date code with yy = year and ww = week.
  - xR = Boot code version and device revision.

Table 1. Device revision table

Revision identifier (R)	Revision description
1A	Initial device revision with Boot ROM version 19.1

2 Errata overview

Table 2. Functional problems table

Functional problems	Short description	Revision identifier	Detailed description
ADC.1	High current consumption in reduced low power modes when using ADC.	'A'	Section 3.1
I^2S.1	FIFO underflow interrupt not generated for I^2S peripheral.	'A'	Section 3.2
I^2S.2	The Most Significant Bit (MSB) of I^2S receive data is forced to 0 if DATALEN is greater than 23.	'A'	Section 3.3
I^2C.1	The AUTOACK feature does not work reliably when the CPU system clock frequency is three times or more than the peripheral clock to the I^2C interface.	'A'	Section 3.4
USART.1	USART receiver timeout feature is not supported.	'A'	Section 3.5
USART.2	The USART receiver idle (RXIDLE) interrupt feature is not supported.	'A'	Section 3.6
SDIO.1	In 4-bit mode, the upper unused data input functions must be selected on GPIO pin.	'A'	Section 3.7
USB.1	In USB high-speed device mode, the NBytes field does not decrement after BULK OUT transfer.	'A'	Section 3.8
USB.2	In USB high-speed device mode, the NBytes field is not correct after BULK IN transfer.	'A'	Section 3.9
USB.3	In USB high-speed device mode, the USB host detects a disconnect during L2 remote wake-up.	'A'	Section 3.10

Table 2. Functional problems table*...continued*

Functional problems	Short description	Revision identifier	Detailed description
USB.4	The L2 remote wake-up signaling is not USB compliant.	'A'	Section 3.11
USB.5	In USB full-speed host mode, linked list on done queue is broken.	'A'	Section 3.12
USB.6	In USB high-speed device and high-speed host modes, the detection handshaking is not working as per USB2.0 specification.	'A'	Section 3.13
USB.7	In USB full-speed device mode, DEV_NEED_CLKST bit in USBCLKSTAT does not go low when LPM token is acknowledged.	'A'	Section 3.14
USB.8	In USB host mode, first ACK not recognized after remote wake-up.	'A'	Section 3.15
USB.9	SE field for an ISO OUT start-split token is wrong when the data length is equal to maximum packet size and when maximum packet size is less than or equal to 188.	'A'	Section 3.16
USB.10	Automatic USB rate adjustment not functional when using multiple hubs.	'A'	Section 3.17
USB.11	A glitch can occur in USB high speed host mode causing host to detect a disconnect.	'A'	Section 3.18
USB.12	USB host port can become disabled when entry into L1 suspend state collides with transmission of any USB token.	'A'	Section 3.19
USB.13	Resetting interrupt endpoint resets DATAx sequence to DATA.1.	'A'	Section 3.20
USB.14	In USB full-speed device mode, the ROOT2 endpoint test fails.	'A'	Section 3.21
USB.15	USB high-speed device in endpoint TX data corruption.	'A'	Section 3.22
SHA.1	Using MEMCTRL after DIGEST Ready to include more blocks via Mastering does not clear DIGEST bit.	'A'	Section 3.23
USB_ROM.1	FRAME_INT is cleared if new SetConfiguration or USB_RESET are received.	'A'	Section 3.24
USB_ROM.2	USB full-speed device fail in the Command/Data/Status Flow after bus reset and bus re-enumeration.	'A'	Section 3.25
IAP EEPROM.1	IAP EEPROM write requires user application to enable SYSCON EEPROM clock before IAP call.	'A'	Section 3.26
USB.16	In USB high-speed device mode, when device isochronous IN endpoint sends a packet of MaxPacketSize of 1024 bytes in response to IN token from host, the isochronous IN endpoint interrupt is not set and the endpoint command/ status list entry for the isochronous IN endpoint is not updated.	'A'	Section 3.27
USB.17	In USB high-speed host mode, only one transaction per micro-frame is allowed for isochronous IN endpoints.	'A'	Section 3.28
PLL.1	P-divider set to 4 could generate the wrong output frequency from the PLL.	'A'	Section 3.29

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
Note.1	For future designs using the USB High-speed feature, it is suggested to use the LPC540xx/LPC54S0xx devices or LPC540LPC54018JxM/LPC54S018JxM.	1A	Section 5.1

3 Functional problems detail

3.1 ADC.1: High current consumption in reduced low power modes when using ADC.

Introduction:

The 12-bit ADC controller is available on all LPC546xx. parts. The ADC can measure the voltage on any of the input signals on the analog input channel. For accurate voltage readings, the digital pin function on the ADC input channel must be disabled by writing a 0 to the DIGIMODE bit in the related IOCON register. This enables the analog mode functionality on the ADC input channel.

Problem:

For applications using the ADC, the current consumption could be higher than expected in reduced power modes (deep-sleep and deep power-down modes) or when the ADC is disabled using the PDRUNCFG register.

Work-around:

To prevent high current consumption, use the following steps in the software:

1. Following a chip reset, all 12 ADC input channels (ADC0_0 to ADC0_11) should be in Digital Mode (DIGIMODE = 1) in the related IOCON registers until the configuration of the ADC block is complete. See the Basic Configuration section in the LPC546xx. 12-bit ADC controller (ADC) chapter of the LPC546xx. User Manual.
2. After configuring the ADC, change only those pins that are used as ADC input channels to Analog Mode (DIGIMODE = 0) in the related IOCON registers before starting ADC conversions.
3. Before entering any reduced power mode (deep-sleep and deep power-down) or before powering down the ADC block (by writing to the PDEN_ADC0 bit in the PDRUNCFG register), the ADC input channel(s) must be changed back to Digital Mode.
4. After waking up from the reduced power mode or when re-enabling the ADC block (PDEN_ADC0 bit in the PDRUNCFG), the software must follow step 2 before starting ADC conversions.

3.2 I^2S.1: FIFO underflow interrupt not generated for I^2S peripheral

Introduction:

Multiple Flexcomm Interfaces are available in the LPC546xx. devices. Flexcomm Interface 6 and Flexcomm Interface 7 can be configured for I^2S peripheral function and the data for all I^2S traffic within one Flexcomm Interface uses the Flexcomm Interface FIFO. During I^2S data transfers, when the transmit FIFO is empty, a FIFO underflow occurs and an interrupt is generated, which is flagged by the UNDERRUN bit in the I^2S FIFOSTAT register.

Problem:

When the FIFO underflow condition occurs, the interrupt from the I^2S peripheral function might not be generated and as a result, the UNDERRUN bit does not get set. This issue does not affect the SPI and USART peripherals.

Work-around:

There is no work-around.

3.3 $I^2S.2$: The Most Significant Bit (MSB) of I^2S receive data is forced to zero if DATALEN > 23

Introduction:

On the LPC546xx devices, the I^2S function is included in Flexcomm Interface 6 and Flexcomm Interface 7. Each of these Flexcomm Interfaces implements one I^2S channel pair. The Data Length (DATALEN) defines the number of data bits to be transmitted or received for all I^2S channel pairs.

Problem:

If the I^2S interface is configured for DATALEN (in I^2S CFG1 register) greater than 23 (25-bit data or greater), the MSB of any received data will be forced to 0. If DATALEN = 24 (25-bit data), bit 24 of received data will always be 0. If DATALEN = 31 (32-bit data), bit 31 of received data will always be 0. The issue occurs regardless of the I^2S operating mode (selected by MODE bits).

Work-around:

There is no work-around.

3.4 $I^2C.1$

Introduction:

In LPC546xx. devices, the I^2C interface has an AUTOACK bit in the Slave Control register. In the slave mode, when this bit is set, it will cause an I^2C header, which matches the slave address SLVADR0 and the direction set by the AUTOMATCHREAD to be ACKed immediately. This is used with the DMA to allow processing of the data without intervention.

Problem:

The AUTOACK feature does not work reliably when the CPU system clock frequency is three times or more than the peripheral clock to the I^2C interface.

Work-around:

The I^2C peripheral clock frequency should be the same or half of the CPU system clock.

3.5 USART.1

Introduction:

A receiver timeout feature for the USART provides a means to get data left for a time in a FIFO that has not reached its threshold to be transferred.

Problem:

The LPC546xx. devices do not support the USART receiver timeout feature.

Work-around:

Timer0 can be used as a USART RX timeout timer and Flexcomm0 as USART0 peripheral in loop back mode. See the technical note TN00013 for more details.

3.6 USART.2

Introduction:

In the USART peripheral, the receiver idle (RXIDLE) interrupt occurs when the RX channel becomes idle.

Problem:

The LPC546xx. devices do not support the USART receiver idle (RXIDLE) interrupt feature.

Work-around:

There is no work-around.

3.7 SDIO.1: In 4-bit mode, the upper unused data input functions must be selected on GPIO pin.

Introduction:

The LPC546xx. devices include a SDIO (Secure Digital I/O) interface that supports SD cards with 1-bit, 4-bit, and 8-bit data mode operations. To operate in the 4-bit mode, SD_D[0] to SD_D[3] functions must be selected on the respective GPIO pins using the IOCON registers (bits 3:0).

Problem:

For the 4-bit mode to work successfully, four otherwise unused upper data bits (SD_D[4] to SD_D[7]) must be functionally assigned to GPIO pins with pull-up resistor. These pins do not need to be physically connected on the hardware.

Work-around:

The following software workaround must be implemented for the 4-bit mode to work. Depending on the package (LQFP208 or TBGA180), signals SD_D[4] to SD_D[7] may be mapped to multiple pins.

For the BGA180 package, program the IOCON registers to select the SD_[D4] to SD_D[7] functions and to enable the on-chip pull-up resistors on the un-bonded GPIO pins:

1. Enable the SD_D[4] function and on-chip pull-up resistor (via IOCON) on pin PIO4_29.
2. Enable the SD_D[5] function and on-chip pull-up resistor (via IOCON) on pin PIO4_30.
3. Enable the SD_D[6] function and on-chip pull-up resistor (via IOCON) on pin PIO4_31.
4. Enable the SD_D[7] function and on-chip pull-up resistor (via IOCON) on pin PIO5_0.
5. Enable SDIO interface to 4-bit mode.

For the LQFP208 package, program the IOCON registers to select the SD_[D4] to SD_D[7] functions and to enable the on-chip pull-up resistors onto 4 unused GPIO pins:

1. Enable the SD_D[4] function and on-chip pull-up resistor (via IOCON) on either pin PIO1_27, PIO3_16, or PIO4_29.
2. Enable the SD_D[5] function and on-chip pull-up resistor (via IOCON) on either pin PIO1_28, PIO3_17, or PIO4_30.
3. Enable the SD_D[6] function and on-chip pull-up resistor (via IOCON) on either pin PIO1_29, PIO3_18, or PIO4_31.
4. Enable the SD_D[7] function and on-chip pull-up resistor (via IOCON) on either pin PIO1_30, PIO3_19, or PIO5_0.
5. Enable SDIO interface to 4-bit mode.

3.8 USB.1: In USB high-speed device mode, the NBytes field does not decrement after BULK OUT transfer

Introduction:

The LPC546xx. device family includes a USB high-speed interface (USB1) that can operate in device mode at high-speed. The NBytes value represents the number of bytes that can be received in the buffer.

Problem:

If the buffer length is less than the maximum packet size and if the application code does not program the maximum packet size, the NByte value is not correct.

Work-around:

Program the NByte to the maximum packet size of that particular endpoint type. The application code must calculate the received number of bytes by subtracting the NByte value from the programming value. The software work-around is implemented on the SDK software platform for the LPC546xx device family.

3.9 USB.2: In USB high-speed device mode, the NBytes field is not correct after BULK IN transfer

Introduction:

The LPC546xx device family includes a USB high-speed interface (USB1) that can operate in device mode at high-speed. When a packet is successfully transferred, the hardware decrements the Nbytes value.

Problem:

The NBytes value is decremented with a wrong value when a packet is successfully transmitted.

Work-around:

There is no work-around. For EP in transfer, the NByte value can be ignored after a packet is transmitted.

3.10 USB.3: In USB high-speed device mode, the USB host detects a disconnect during L2 remote wake-up

Introduction:

The LPC546xx device family includes a USB high-speed interface (USB1) that can operate in device mode at high-speed. The USB interface goes into L2 suspend state if there is no activity in the USB bus for more than 3 ms and can wake up if there is transmission from the host or via the device's software initiated remote wake-up.

Problem:

When the LPC546xx device sends a L2 remote wake-up, an unexpected signal is generated on the bus, which makes the USB host port to detect a disconnect.

Work-around:

To continue USB operation after L2 remote wake-up, the USB host must reset its port and the LPC546xx device will be re-enumerated by the host.

3.11 USB.4: The L2 remote wake-up signaling is not USB compliant

Introduction:

The USB interface on the LPC546xx device family is USB certified by the USB-IF.

Problem:

The L2 remote wake-up feature is an optional feature and was not part of the USB compliance testing.

Work-around:

There is no work-around.

3.12 USB.5: In USB full-speed host mode, linked list on done queue is broken

Introduction:

The LPC546xx device family includes a USB full-speed interface (USB0) that can operate in host mode at full-speed. The completed TD must go on the Done queue.

Problem:

The NextTD field of a General Transfer Descriptor is not updated with the value of the HcDoneHead register when the ConditionCode is not equal to 0x0 or 0x9.

The NextTD field of an Isochronous Transfer Descriptor is not updated with the value of the HcDoneHead register when the ITD is completed.

Work-around:

The following work-arounds must be implemented for this problem.

1. Ignore the unused TD on the Done Queue. Use the SDK library to implement software work-around.
2. Limiting the hub interrupt endpoint interval to be a maximum of twice the smallest value of all other interrupt or isochronous endpoints.

3.13 USB.6: In USB high-speed device and high-speed host modes, the detection handshaking does not work per the USB2.0 specification

Introduction:

See the USB2.0 specification for details regarding the USB High-speed Detection Handshake protocol.

Problem:

In USB high-speed device and host modes, the detection handshake fails for the following conditions:

Condition 1:

As a high-speed device, the LPC546xx does not see the HOST KJ sequence and as a result, it does not recognize whether the HOST can support high-speed. Therefore, the LPC546xx will behave only as a full-speed device instead of a high-speed device.

Condition 2:

As a high-speed host, the LPC546xx does not see the Device K and as a result, it does not send the KJ sequence to the Device. Therefore, the LPC546xx will only behave as a full-speed host instead of a high-speed host.

Work-around:

For condition 1, the software work-around is implemented on the SDK software platform for the LPC546xx device (implemented in usb_dev_hid_mouse only (API called "USB_DeviceHsPhyChirpIssueWorkaround0") and must be used to avoid this issue.

For condition 2 (workaround is not implemented for the SDK host example), the application software must use one of the timer peripherals (Multi-rate timer, Repetitive Interrupt Timer, CTIMERS0-4) and use the following sequence:

1. Wait until a Port Connect interrupt is received.
2. Poll PSPD field (bits 21 to 20) in PORTSC1 register. If PSPD not equal to 00b, then wait 100 ms to comply with section 7.1.7.3 of the USB 2.0 specification.
3. Write 0011b to PTC field (bits 19 to 16) in PORTSC1 register (start of USB bus reset – SE0) and wait 250 µs.
4. Start 7 ms timer.
5. Poll linestate field in PORTSC1 register (polls 3 times with 10 µs delay) until timer times-out or the following condition is TRUE:

5.1.0: If (all 3) linestate not equal to 00b, then (HS device attached):

5.1.1: Stop timer and restart 4 ms timer.

5.1.2: Poll linestate field in PORTSC1 register (polls 3 times with 2.5 µs delay) until timer times-out or the following condition is TRUE:

5.1.2.1: If (all 3) linestate is equal to 00b then:

a) Write 0010b to PTC field (bits 19 to 16) in PORTSC1 register and wait 50 µs.

b) Write 0001b to PTC field and wait 50 µs.
c) Repeat steps a and b 40 times.
d) Write 0011b to PTC field and wait 200 µs.
e) Write 0101b to PTC field.
5.1.3 Schedule GetDeviceDescriptor request and wait on completion.
5.1.4 Write 0000b to PTC field and wait at least 5 µs.
5.1.5 Initiate a USB bus reset by writing 1b to PR bit in PORTSC1 register.
5.2.0 If 7 ms timer times-out because linestate is continuously 00b, then (FS device attached),
5.2.1 Wait an additional 4 ms.
5.2.2 Write 0000b to PTC field.
5.2.3 Schedule GetDeviceDescriptor request and initiate a USB bus reset.

3.14 USB.7: In USB full-speed device mode, DEV_NEED_CLKST bit in USBCLKSTAT does not go low when LPM token is acknowledged.

Introduction:

The LPC546xx device family includes a USB full-speed interface (USB0) that can operate in device mode at full-speed. When used in L2 suspend mode, the NEEDCLK signal goes LOW (USB0CLKSTAT register) and can be used as an indicator (USB0CLKSTAT register) to reduce power by disabling the clock to the USB peripheral.

Problem:

In L1 suspend mode, the NEEDCLK signal does not go LOW, which prevents the ability to disable the clock to the USB interface.

Work-around:

There is no work-around.

3.15 USB.8: In USB host mode, first ACK is not recognized after remote wake-up

Introduction:

The LPC546xx device family includes a USB interface (USB1) that can operate in host mode. The USB host interface features L2 suspend state and remote wake-up acknowledgement from device.

Problem:

The LPC546xx (using port USB1) as a USB host does not recognize the first ACK after remote wake-up from device and as a result, halts the PTD.

Work-around:

After the PTD is halted, the LPC546xx host should re-schedule the SETUP transaction a second time to resume USB communication.

3.16 USB.9: SE field for an ISO OUT start-split token is wrong when the data length is equal to maximum packet size and when maximum packet size is less than or equal to 188

Introduction:

The LPC546xx device family includes a USB high-speed interface (USB1) that can operate in host mode at high-speed. The LPC546xx as a high-speed USB host must send an ISO OUT start-split transaction where the data length ≤ 188 bytes or when the data length is equal to maximum packet size. This transaction must have the SE field set to ALL (=11b).

Problem:

When the LPC546xx as a high-speed USB host sends an ISO OUT start-split token and the Number of Bytes to Transfer is equal to the MaxPacketSize, the SE field in the token is set to incorrect value (BEGIN = 10b).

Work-around:

Software must program the MaxPacketLength field in the PTD to the value reported in the siTD + 1. The software work-around is implemented on the SDK software platform for the LPC546xx device family.

3.17 USB.10: Automatic USB rate adjustment is not functional when using multiple hubs

Introduction:

Full-speed and low-speed signaling uses bit stuffing throughout the packet without exception. If the receiver sees seven consecutive ones anywhere in the packet, then a bit stuffing error has occurred, and the packet should be ignored.

The time interval just before an End of Packet (EOP) is a special case. The last data bit before the EOP can become stretched by hub switching skews. This is known as dribble and can lead to a situation where dribble introduces a sixth bit that does not require a bit stuff. Therefore, the receiver must accept a packet where there are up to six full bit times at the port with no transitions prior to the EOP.

Problem:

The LPC546xx devices use the start of an EOP for frequency measurements. This is not functional when going through multiple hubs that introduce a dribble bit because of hub switching skews. For this reason, the start of the EOP cannot be used for frequency measurements for automatic USB rate adjustment (by setting USBCLKADJ in FROCTRL register). The problem does not occur when a single hub is used.

Work-around:

Use the FRO calibration library provided in TN00032. This library allows the application to have a crystal-less USB device operation in full-speed mode.

3.18 USB.11: In USB high speed host mode, a glitch can occur causing host to detect a disconnect

Introduction:

The LPC546xx includes a USB High Speed interface (USB1) that can operate in host mode at high speed. The USB high speed host interface features L1 and L2 suspend state. A device can be put in L1 suspend mode (when L1 is supported by the device) when the host controller (LPC546xx) generates an LPM Token to enter the L1 state. A host controller (LPC546xx) can put the device in L2 suspend mode by keeping the USB lines idle for more than 3 milliseconds. A suspended device wakes up if there is resume signaling from the host (host-initiated wake-up).

Problem:

When the LPC546xx (high speed host) is connected to a high speed device, at the end of a resume a glitch can occur on the USB lines. When this glitch is generated, the USB HS host detects a disconnect and the USB host port is disabled.

Work-around:

There is no software work-around to prevent the host disconnect.

The software can generate a USB bus reset after the host detects a disconnect and a reconnect by the device (CCS and CSC bits both set to 1), followed by a re-enumeration of the USB device. The software work-around is implemented on the SDK software platform for the LPC546xx.

3.19 USB.12: The USB host port can become disabled when entry into L1 suspend state collides with transmission of any USB token.

Introduction:

The LPC546xx includes a USB high speed interface (USB1) that can operate in host mode at high speed. The USB high speed host interface features L1 suspend state. A device can be put in L1 suspend mode (when L1 is supported by the device) when the host controller (LPC546xx) generates an LPM Token to enter the L1 state. A host controller will put a device in L1 suspend state if it has no pending transactions to the USB device.

Problem:

When the LPC546xx (high speed host) is programmed to send out the LPM Token at the same time when it needs to transmit another USB token, the port can become disabled.

Work-around:

The software work-around is to make sure that there are no PTD structures on the asynchronous or periodic lists with the valid bit set to 1b. It must also make sure that setting the USB port on the host controller in L1 suspend mode is done at the beginning of the (micro-) frame. For this, the USB software can wait until a Start-of-Frame interrupt is received to program the port entering L1 suspend mode.

3.20 USB.13: Resetting interrupt endpoint resets DATAx sequence to DATA.1

Introduction:

The LPC546xx includes a USB High Speed interface (USB1) that can operate in device mode at high speed. The T bit in command/status list determines if the endpoint type is generic endpoint or periodic endpoint. When the endpoint type is set to periodic, the RF/TV bit in command/status list determines if the endpoint type is isochronous endpoint or interrupt endpoint. When the TR bit in command/status list is set to ‘1’, the toggle value will set to the value indicated in the RF/TV bit.

Problem:

When the endpoint type is set to interrupt with T bit as ‘1’ and RF/TV bit as ‘1’, the data toggle for the interrupt endpoint with TR bit as ‘1’ resets to DATA1.

Work-around:

For applications that have strict requirements of the data toggle value, the following is the the software work-around:

1. Set INTONNAK_AO and INTONNAK_AI bits to ‘1’ in the Device Command/Status register.
2. Set A=0, TR=1, RF/TV=0, T=0 (this will force the device to return a NAK handshake and reset the internal toggle value to zero).
3. Wait until an interrupt is received. Read the Endpoint Toggle register and check the value of the endpoint toggle. If the toggle is reset to ‘0’, then go to step 4 else wait for the next interrupt.
4. Set A=1, TR=0, RF/TV=1, T=1 (the endpoint is back to the normal operation)

The result of this work-around is when an endpoint is reset, a NAK handshake is returned on the first received token.

3.21 USB.14: In USB full-speed device mode, the ROOT2 endpoint test fails

Introduction:

The LPC546xx includes a USB full speed interface (USB0) that can operate in device mode at full speed. It supports 10 physical (5 logical) endpoints including control endpoints. The device should not respond to those endpoints which are not supported.

Problem:

The device NAKed the OUT token addressed to an endpoint that is not present on the device causing the ROOT2 endpoint test to fail.

Work-around:

There is no work-around.

3.22 USB.15: USB high-speed device in endpoint TX data corruption

Introduction:

The LPC546xx includes a USB high-speed interface (USB1) that can operate in device mode at high-speed. The endpoint type can be configured as control, interrupt, bulk, and isochronous to their corresponding maximum packet sizes of 64, 1024, 512, and 1024 bytes.

Problem:

For all endpoint types and TX (IN) transfer, the first byte of the transfer data is changed to 0 if all the following conditions are met:

- A TX (IN) transfer happens after a RX (OUT) transfer.
- The RX (OUT) transfer length is 4 + N*16 (N ≥ 0) bytes.

The TX (IN) transfer and the RX (OUT) transfer can be on either the same endpoint or different endpoint.

Work-around:

A NAKed Tx (IN) transfer in-between the RX (OUT) and TX (IN) transfers solves the issue.

3.23 SHA.1: Using MEMCTRL after DIGEST Ready to include more blocks via Mastering does not clear DIGEST bit.

Introduction:

The LPC546xx includes a SHA hash block to compute SHA1 and SHA2-256 hash digests on flash images or messages in RAM. For maximum performance and ease of use, the hash block includes a master on the internal buses of the chip to read multiple blocks of memory while hashing, without involvement of the processor. This mastering model permits hashing up to 128 K bytes of memory (Flash, RAM, or SPI Flash).

Problem:

If the application uses the mastering on up to 128 K bytes and then uses it for additional blocks (without starting new), the DIGEST (digest ready) status does not clear when starting the next sequence via mastering. If the processor or DMA is used for the additional blocks, the DIGEST status is cleared.

Work-around:

If the purpose for the additional block(s) is to hash the last block (with padding and length), then the processor or DMA may be used to write the 16 words via INDATA, and the DIGEST status will clear when the 1st word is written.

If the purpose for additional blocks is to do a large number of blocks (for example, after doing 128 K, another 64 K is to be hashed), then the 1st block may be started by the processor (that is, the processor writes the 16 words to INDATA) followed by configuring MEMADDR and MEMCTRL for the remaining blocks. The MEMCTRL should be written within 64 cycles of writing the last word to INDATA to ensure DIGEST is 0.

3.24 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);
}
```

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

Introduction:

The LPC546xx 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);
```

3.26 IAP EEPROM.1: IAP EEPROM write requires user application to enable SYSCON EEPROM clock before IAP call

Introduction:

All LPC546xx devices include ROM-based services for programming and reading the flash memory in addition to other functions. In-Application (IAP) programming performs erase and write operation on the on-chip flash memory as directed by the end-user application code. LPC546xx supports IAP Write EEPROM page command that writes given data in the provided memory to a page of EEPROM

Problem:

The LPC546xx IAP EEPROM Write page API does not enable the EEPROM clock for the write operation. This causes the EEPROM write operation to fail when using the IAP API. This problem does not occur for IAP Read EEPROM page command.

Work-around:

Enable the EEPROM clock (set bit 9) in AHBCLKCTRL0 register (offset 0x200) of SYSCON before calling the IAP EEPROM write function.

3.27 USB.16: In USB high-speed device mode, when device isochronous IN endpoint sends a packet of MaxPacketSize of 1024 bytes in response to IN token from host, the isochronous IN endpoint interrupt is not set and the endpoint command/status list entry for the isochronous IN endpoint is not updated

Introduction:

The LPC546xx device family include a USB high-speed interface (USB1) that can operate in device mode at high-speed. The isochronous IN endpoint supports a MaxPacketSize of 1024 bytes.

Problem:

When device isochronous IN endpoint sends a packet of MaxPacketSize of 1024 bytes in response to IN token from host, the isochronous IN endpoint interrupt is not set and the endpoint command/status list entry for the isochronous IN endpoint is not updated.

Work-around:

Restrict the isochronous IN endpoint MaxPacketSize to 1023 bytes in device descriptor.

3.28 USB.17: In USB high-speed host mode, only one transaction per micro-frame is allowed for isochronous IN endpoints

Introduction:

The LPC546xx device family include a USB high-speed interface which can operate in host mode. Up to three high-speed transactions are allowed in a single micro-frame to support high-bandwidth endpoints. This mode is enabled by setting the Mult (Multiple) field in the Proprietary Transfer Descriptor (PTD) and is used to indicate to the host controller the number of transactions that should be executed per micro-frame. The allowed bit settings are:

00b Reserved. A zero in this field yields undefined results.

01b One transaction to be issued for this endpoint per micro-frame.

10b Two transactions to be issued for this endpoint per micro-frame.

11b Three transactions to be issued for this endpoint per micro-frame.

Problem:

For High-bandwidth mode, using multiple packets (MULT = 10b or 11b) in a frame causes unreliable operation. Only one transaction (MULT = 01b) can be issued per micro-frame.

Work-around:

There is no software workaround. Only one transaction can be issued per micro-frame.

3.29 PLL.1: P-divider set to 4 could generate the wrong output frequency from the PLL

Introduction:

On the LPC546xx PLL, the Fcco frequency must be either the actual desired output frequency, or the desired output frequency times 2 x P, where P is range from 1 to 32 (2^5). The Fcco frequency must also be a multiple of the PLL reference frequency, which is either the PLL input, or the PLL input divided by N, where N is from 2 to 256.

Problem:

The P-divider when set for divide by 4 mode can erroneously arrive in divide by 2 mode. The high frequency spikes coming from the level shifter during startup of the PLL can cause the P-divider to jump into the wrong division state only when set in divide by 4 mode. This issue affects both the System PLL and Audio PLL.

Work-around:

Use other P values other than 4 to achieve desired output frequency. P = 1 – 32 and P ≠ 4.

4 AC/DC deviations detail

No known errata.

5 Errata notes

5.1 Note.1.

Due to the USB erratas listed above, for future designs using the USB High-speed feature, it is recommended to use LPC540xx/LPC54S0xx devices or the LPC54018JxM/LPC54S018JxM devices since the USB issues have been fixed.

6 Revision history

Table 5. Revision history

Document ID	Release date	Description
ES_LPC546xx v. 2.4	25 January 2024	• Added Section 3.29
ES_LPC546xx v. 2.3	23 April 2021	• Added USB.16. • Added USB.17.
ES_LPC546xx v. 2.2	22 October 2019	• Added Section 5.1 and additional clarifications.
ES_LPC546xx v. 2.1	23 October 2018	• Added USB.15. • Added IAP EEPROM.1
ES_LPC546xx v. 2.0	13 March 2018	• USB.14. • Changed title of Section 3.23 to Section 3.23Ready to include more blocks via Mastering does not clear DIGEST bit.".
ES_LPC546xx v. 1.9	7 March 2018	• USB_ROM.2.
ES_LPC546xx v. 1.8	26 February 2018	• Updated work-around for Section 3.17functional when using multiple hubs".
ES_LPC546xx v. 1.7	9 November 2017	• Added LPC54605J256BD100, LPC54605J512BD100, LPC54605J256 ET100, LPC54605J512ET100 part numbers in Keywords. • Added USB.13.
ES_LPC546xx v. 1.6	4 August 2017	• USB_ROM.1.
ES_LPC546xx v. 1.5	26 May 2017	• Added part number LPC54628J512ET180. • Added SHA.1.
ES_LPC546xx v. 1.4	5 May 2017	• Updated Table 2: changed heading row to Functional problems.
ES_LPC546xx v. 1.3	20 April 2017	• Added USB.11. • Added USB.12. • Added 100-pin devices. • Updated Section 1.
ES_LPC546xx v. 1.2	6 March 2017	• Renamed title to LPC546xx. • Updated work-around for USB.10: Use the external crystal instead of the FRO as a clock source to the PLL.
ES_LPC546xx v. 1.1	24 February 2017	• Removed S parts.
ES_LPC546xx v. 1	15 December 2016	• Initial version.

7 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

8 Legal information

8.1 Definitions

Draft — A draft status on a document indicates that 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 in a draft version of a document and shall have no liability for the consequences of use of such information.

8.2 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.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

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.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

8.3 Trademarks

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

NXP — wordmark and logo are trademarks of NXP B.V.

Contents

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

Date of release: 25 January 2024
Document identifier: ES_LPC546xx