ES_LPC436x Flash

Errata sheet LPC436x flash-based devices

Rev. 2.3 — 24 September 2019 Errata sheet

Document information

Info	Content
Keywords	LPC4367JET256; LPC4367JBD208; LPC4367JET100; ARM Cortex-M4 flash-based devices, Rev A, B 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
2.3	20190924	• Added Rev B. • Updated for device revision with copper wire conversion.
2.2	20180307	• USBROM.3
2.1	20160511	• Added RTC.1.
2.0	20151210	• Added PMC.1.
1.0	20151027	• 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 LPC436x flash-based devices (hereafter referred to as 'LPC43xx') typically have the following top-side marking:

LPC43xxxxxxxx

xxxxxxxx

xxxYYWWxR[x]

Field 'YY' states the year the device was manufactured. Field 'WW' states the week the device was manufactured during that year. Field x before the field 'R' is the boot ROM version. The last/second to last letter in the last line (field 'R') identifies the device revision.

This Errata Sheet covers the following revisions of the LPC43xx flash-based devices:

Table 1. Device revision table

Revision identifier (R)	Revision description
'B'	Second device revision
'A'	Initial device revision

2. Errata overview

Table 2. Functional problems table

Functional problems	Short description	Revision identifier	Detailed description
EMC.1	Operating frequency of EMC lower than data sheet value.	'A', 'B'	Section 3.1
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', 'B'	Section 3.2
SRAM.1	SRAM in deep sleep and power down modes may lose state.	'A', 'B'	Section 3.3
USB.1	USB0 unable to communicate with low-speed USB peripheral in host mode when using full-speed hub.	'A', 'B'	Section 3.4
USB.2	The USB_SOF_Event may fire earlier than expected and/or a false interrupt may be generated.	'A', 'B'	Section 3.5
USBROM.1	Nested NAK handling of EP0 OUT endpoint.	'A', 'B'	Section 3.6
USBROM.2	Isochronous transfers.	'A', 'B'	Section 3.7
USBROM.3	USB full-speed device fail in the Command/Data/Status Flow after bus reset and bus re-enumeration.	'A', 'B'	Section 3.8
SD/MMC.1	Data CRC error returned on CMD6 command.	'A', 'B'	Section 3.9
RESET.1	Master Reset (MASTER_RST) and M4 Reset (M4_RST) are not functional.	'A', 'B'	Section 3.10

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

Functional problems	Short description	Revision identifier	Detailed description
RESET.2	Loss of device functionality on reset via nRESET in deep-sleep and power-down mode.	'A', 'B'	Section 3.11
PMC.1	Wake-up from deep-sleep mode using USB0/1 peripherals.	'-'	Section 3.12
RTC.1	The Real Time Clock (RTC) does not work reliably when there is I/O switching activity on pins near to the RTCX1 oscillator input pin.	'A', 'B'	Section 3.13

Table 3. AC/DC deviations table

AC/DC deviations	Short description	Product version(s)	Detailed description
n/a	n/a	n/a	n/a

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 EMC.1

Introduction:

The LPC436x parts contain an External Memory Controller (EMC) capable of interfacing to external SDRAM, SRAM, and asynchronous parallel flash memories. The EMC can be configured to operate at the processor core frequency (BASE_M4_CLOCK) or the core frequency divided by 2.

Problem:

For SDRAM, the electrical characteristic of the LQFP208 package limits the operating frequency of the EMC to a certain level, which is lower than the specified value in the data sheet. Choosing an SDRAM clock of 72MHz as the upper limit provides some safety margin. This frequency is either achieved by a core and EMC frequency of 72MHz, or by a 144MHz core and a 72MHz EMC frequency. However, SDRAM performance can vary depending on board design and layout.

Work-around:

There is no work-around.
The upper limit of the SDRAM clock frequency is highly dependent on the PCB layout and the quality of the power supply and de-coupling circuitry.

3.2 I2C.1

Introduction:

The I2C monitor allows the device to monitor the I2C traffic on the I2C 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;
```

3.3 SRAM.1

Introduction:

SRAM state is retained in deep sleep and power down modes.

Problem:

Incorrect settings may lead to SRAM state retention loss over time and temperature. This can cause erratic behavior due to SRAM data loss after wake-up from deep sleep mode or power down mode.

Work-around:

Reserved register at 0x40043008 bits 17:16 should be set to 0x2 before entering deep sleep mode or power down mode.

```
#define CREG0_008       (0x40043008)
#define PD0_SLEEP0_MODE (0x4004201c)

#define PMC_PWR_DEEP_SLEEP_MODE 0x3F00AA
#define PMC_PWR_POWER_DOWN_MODE 0x3FFCBA

unsigned int regval;

// EXAMPLE 1:
regval = *((unsigned int *) CREG0_008);
regval |= (1 << 17);
regval &= ~(1 << 16);
*((unsigned int *) CREG0_008) = regval;

// prepare for entering deep sleep
*((unsigned int *) PD0_SLEEP0_MODE)  = PMC_PWR_DEEP_SLEEP_MODE;

// enter deep sleep
__wfi();

// EXAMPLE 2:
regval = *((unsigned int *) CREG0_008);
regval |= (1 << 17);
regval &= ~(1 << 16);
*((unsigned int *) CREG0_008) = regval;

// prepare for entering power down
*((unsigned int *) PD0_SLEEP0_MODE)  = PMC_PWR_POWER_DOWN_MODE;

// enter power down
__wfi();
```

3.4 USB.1

Introduction:

The LPC436x 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.

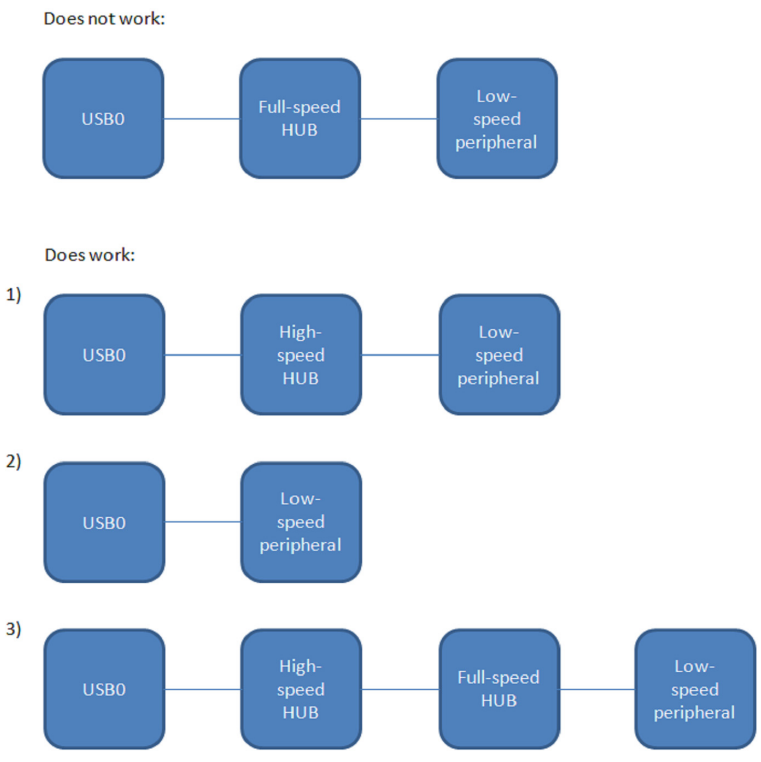

Fig 1. Suggested USB0 to low-speed peripheral connections

3.5 USB.2

Introduction:

The LPC436x flash-based devices contain an event handler for USB SOF detection from the host called the USB_SOF_Event. When it is enabled this event fires at the start of each USB frame, once per millisecond in full-speed mode or once per 125 microseconds in high-speed mode, and is synchronized to the USB bus.

Problem:

The USB_SOF_Event may fire earlier than expected and/or an additional (false) interrupt may be generated.

Work-around:

There is no work-around. The USB_SOF_Event cannot be used in full-speed and high-speed device mode in case the system needs an interrupt that is aligned with the incoming SOF tokens.

3.6 USBROM.1

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;
}
```

3.7 USBROM.2

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

3.8 USB_ROM.3

Introduction:

The LPC436x 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);
```

3.9 SD/MMC.1

Introduction:

The LPC436x parts have the SD/MMC interface. After power up, the SD memory card is in the default speed mode, and by using the Switch Function command (CMD6), the Version 1.10 and higher SD memory cards can be placed in High-Speed mode. In response to the CMD6 command, the SD card returns a 512-bit block of data containing the available features and actual settings. The SDIO interface is setup for 4-bit data and therefore, the 512 bits are returned on the four data lines in 128 clocks followed by 16 clocks of CRC data.

Problem:

The CMD6 returned status block always gets a data CRC error although the status data is correct. The data CRC error prevents the switching of SD memory card from the default mode to High-Speed mode.

Work-around:

To capture the 512 bits of data and CRC data, the DMA buffer length and SD/MMC BYTCNT are increased to 72 and then the CRC is calculated in software. If the CRC is correct for all four data lines, the error is cleared.

3.10 RESET.1

Introduction:

The LPC436x parts contain a Reset Generation Unit (RGU) that generates various resets; Core Reset (CORE_RST), Peripheral Reset (PERIPH_RST), Master Reset (MASTER_RST), and M4 Reset (M4_RST).

Problem:

On the LPC436x, MASTER_RST and M4_RST are not functional.

Work-around:

There is no work-around. To reset the entire chip use the CORE_RST instead of using MASTER_RST or M4_RST.

3.11 RESET.2

Introduction:

The LPC436x devices are initialized after a reset. If a reset occurs via nRESET pin when the part is in deep-sleep or power-down mode, the initialization state of the device may be erroneous and some functionality of the device may be lost.

Problem:

When the part is in deep-sleep or power-down mode and if an external reset occurs via nRESET pin being activated, as the part comes out of reset, the reset state of some functional blocks may be incorrect. This may result in loss of functionality of the device. The actual functionality lost may vary from part to part depending on the erroneous reset state of the functional blocks. The possible affected blocks are: Ethernet, LCD controller, CAN0, CAN1, USB0, USB1, SGPIO, AES, Cortex-M0 coprocessor and Cortex-M0 subsystem (if present), 12-bit ADC, SRAM size at 0x2000 0000 may change to 16 kB, SRAM size at 0x2000 8000 may change to 0 kB, and SRAM size at 0x2000 C000 may change to 0 kB.

Work-around:

There are two possible work-arounds:

1. In the application software, before initializing peripherals, the code should assert a soft reset using the following steps:
   a. Read the value in power-down modes register (PD0_SLEEP0_MODE).
   b. If the value in the PD0_SLEEP0_MODE0 register represents deep-sleep mode or power-down mode, then the user should check if a reset event occurred on the nRESET pin (bit ‘19’ in the Event Status register).
   c. If the reset event occurred, the software should set the PD0_SLEEP0_MODE register to deep power-down mode and assert a soft reset using the CORE_RST (bit ‘0’ in the RESET_CTRL0 register).

```
/* Check if wake up event happens in deep Sleep or power Down mode */
    if((LPC_PMC->PD0_SLEEP0_MODE == PMC_PWR_DEEP_SLEEP_MODE)
        || (LPC_PMC->PD0_SLEEP0_MODE == PMC_PWR_POWER_DOWN_MODE))
    {
        /* Check if the wake up event is due to nRESET pin in Event router */
        if(LPC_EVRT->STATUS & (1<<19))
        {
            /* Set power state in PMC */
            LPC_PMC->PD0_SLEEP0_MODE = PMC_PWR_DEEP_POWER_DOWN_MODE;
            /* Set CORE_RST in RGU */
            LPC_RGU->RESET_CTRL0 = (1<<0);
        }
    }
```

2. To initialize the device correctly, assert a second external reset signal to the nRESET pin after 20 μs from the first reset.

3.12 PMC.1

Introduction:

On the LPC43xx devices, USB0 and USB1 peripherals can act as wake-up sources in Sleep mode and in Deep-sleep mode (by setting bits 9 and 10 in CREG1 register).

Problem:

The LPC43xx Rev '-' devices do not support wake-up from Deep-sleep mode using USB0 and USB1. The USB0 and USB1 peripherals can wake-up these devices only from Sleep mode.

Work-around:

There is no work-around.

3.13 RTC.1

Introduction:

The Real Time Clock (RTC) is a set of counters for maintaining a time base when system power is off, and optionally when it is on. The RTC block is designed to consume very little power, using an external 32.768 kHz crystal to generate a 1 Hz internal time reference. The RTC is powered by its own power supply pin, VBAT.

Problem:

On the LPC43xx devices, when there is I/O switching activity on pins close to the RTCX1 pin, the RTC does not work reliably due to noise coupling into the 32.768 kHz oscillator circuit design. This results in additional (spurious) clock cycles for the counters and therefore in a time shift of the RTC.

On the LQFP144 package, I/O switching activity on pins P3_7 (pin number 123) and P3_8 (pin number 124) can cause noise coupling into the RTCX1 oscillator input pin (pin number 125).

On the LQFP208 package, I/O switching activity on pins PB_4 (pin number 180) and PB_5 (pin number 181) can cause noise coupling into the RTCX1 oscillator input pin (pin number 182).

Work-around:

1. For both LQFP packages, the pins adjacent to RTCX1 can be avoided since the functions on these pins are multiplexed on other pins. However, if using the SPIFI interface with the LQFP144 package, there are no alternative pins which have SPIFI functions. In that case, apply work-around 2.
2. If an on-chip 32.768 kHz oscillator is used, the RTCX1 pin will be sensitive to noise from the adjacent pins. Use an external 32.768 kHz clock source (from a host system or from an external oscillator) as an input to the RTCX1 pin to avoid noise coupling. This work-around is valid for both LQFP package types. See the application information section in the data sheet for more information on using an external clock.

4. AC/DC deviations detail

N/A

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: 24 September 2019
Document identifier: ES_LPC436X_FLASH