1 Introduction

The LPC51U68 based on Arm® Cortex®-M0+ is a low-cost, low-power consumption, 32-bit Micro Controller Unit (MCU) family. It operates at frequencies of up to 100 MHz and supports up to 256 KB on-chip flash memory and up to 96 KB total SRAM composed of up to 64 KB main SRAM, plus an additional 32 KB SRAM. The on-chip peripherals in LPC51U68 includes one DMA controller, 48 General-Purpose I/O (GPIO) pins, one CRC engine, one 12-bit ADC, one 32-bit Real-Time Clock (RTC), one multiple-channel Multi-Rate 24-bit Timer (MRT), one Windowed WatchDog Timer (WWDT), eight Flexcomm interfaces which can be selected by software to be a USART, SPI, or I²C interface.

The LPC51U68 I²C bus contains the following features:

- Supporting the following bus speeds:
  - Standard mode, up to 100 kbits/s
  - Fast-mode, up to 400 kbits/s
  - Fast-mode Plus, up to 1 Mbits/s (on pins PIO0_23 and 24 or PIO0_25 and 26 that include specific I²C support)
  - High speed mode, 3.4 Mbits/s as a Slave only (on pins PIO0_23 and 24 or PIO0_25 and 26 that include specific I²C support)

- Supporting both Multi-master and Multi-master with Slave functions

- Supporting multiple I²C slave addresses in the hardware

- Qualifying one slave address with a bit mask or an address range to respond to multiple I²C bus addresses

- 10-bit addressing supported with software assistance

- Supporting System Management Bus (SMBus).

- Separated DMA requests for Master, Slave, and Monitor functions

- Waking up the device from deep-sleep mode, as no chip clocks are required to receive and compare an address as a Slave

- Automatic modes optionally allow less software overhead for some use cases

This document describes how to trigger and detect I²C transmission errors including Start/Stop error, Arbitration Loss, SCL time-out, Event time-out on LPC51U68. I²C transmission errors are triggered by introducing external glitch which is also called as interference.


2 I2C bus introduction

I2C is an 8-bit, bidirectional, serial multi-master bus. It supports four modes including:

- Standard mode up to 100 kbit/s
- Fast mode up to 400 kbit/s
- Fast mode Plus up to 1 Mbit/s
- High speed mode up to 3.4 Mbit/s

As shown in Figure 2, I2C uses a Serial Clock Line (SCL) and a Serial Data Line (SDA) for data transfer. The SCL and SDA of master devices and slave devices are connected to the SCL and SDA of I2C bus. For more details about I2C bus specification, see I2C-bus specification and user manual (document UM10204).

![Multi-Master I2C bus architecture](image)

The output stages of devices connected to the bus must have an open-drain or open-collector to perform the wired-AND function. The master device initiates a transfer, generates clock signal, addresses the slave device, and terminates a transfer. Figure 3 illustrates the timing of I2C bus data transmission.
As shown in Figure 4, I2C transactions begin with a START (S) and are terminated by a STOP (P). A HIGH to LOW transition on the SDA line while SCL is HIGH defines a START condition. A LOW to HIGH transition on the SDA line while SCL is HIGH defines a STOP condition.

### 3 System overview

As shown in Figure 5, the system described in this document consists of three parts including one master, one slave and one glitch generator.

The master and slave perform normal data transfer. They are implemented by two LPCXpresso51U68 (OM40005) evaluation boards.

The glitch generator is used to interfere with normal I2C data transmission, resulting in transmission errors including Start/Stop error, Arbitration Loss, SCL time-out, Event time-out. The glitch generator is implemented by LPCXpresso55s69 (Revision A) evaluation board.
The glitch generator is implemented by the state machine generated by LPC55S69 SCTimer.

NOTE
The glitch generator detects the serial clock on the SCL line and applies glitch on the SCL line or the SDA line according to the type of I2C transmission errors.

For specific details of I2C transmission errors, see I2C transmission error type.
4 Hardware configurations

Table 1, Table 2, and Table 3 describe the detailed hardware configurations of I2C master, I2C slave and glitch generator.

Table 1. I2C Master hardware configurations

<table>
<thead>
<tr>
<th>Part</th>
<th>Port</th>
<th>Pin</th>
<th>Mark on Evk board</th>
<th>Alternative function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPC51U68</td>
<td>0</td>
<td>25</td>
<td>J1-1</td>
<td>FC4_RTS_SCL_SSEL1</td>
<td>SCL line</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>26</td>
<td>J1-3</td>
<td>FC4_CTS_SDA_SSEL0</td>
<td>SDA line</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>29</td>
<td>J2-5</td>
<td>PIO0_29</td>
<td>Start/Stop error indicator LED</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>2</td>
<td>J8-1</td>
<td>PIO0_2</td>
<td>Event time-out indicator LED</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>3</td>
<td>J8-3</td>
<td>PIO0_3</td>
<td>other errors indicator LED</td>
</tr>
</tbody>
</table>

Table continues on the next page...
Table 1. \(\text{I}^2\text{C}\) Master hardware configurations (continued)

<table>
<thead>
<tr>
<th>Part</th>
<th>Port</th>
<th>Pin</th>
<th>Mark on Evk board</th>
<th>Alternative function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>J8-5</td>
<td>PIO1_9</td>
<td>SCL time-out indicator LED</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>J8-8</td>
<td>PIO1_10</td>
<td>Arbitration Loss indicator LED</td>
<td></td>
</tr>
</tbody>
</table>

Table 2. \(\text{I}^2\text{C}\) Slave hardware configurations

<table>
<thead>
<tr>
<th>Part</th>
<th>Port</th>
<th>Pin</th>
<th>Mark on Evk board</th>
<th>Alternative function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPC51U68</td>
<td>0</td>
<td>25</td>
<td>J1-1</td>
<td>FC4_RTS_SCL_SSEL1</td>
<td>SCL line</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>26</td>
<td>J1-3</td>
<td>FC4_CTS_SDA_SSEL0</td>
<td>SDA line</td>
</tr>
</tbody>
</table>

Table 3. \(\text{I}^2\text{C}\) Glitch generator hardware configurations

<table>
<thead>
<tr>
<th>Part</th>
<th>Port</th>
<th>Pin</th>
<th>Mark on Evk board</th>
<th>Alternative function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPC55S69</td>
<td>0</td>
<td>5</td>
<td>P17-8</td>
<td>SCT0_GPI5</td>
<td>Detect clock on SCL line</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>3</td>
<td>P17-11</td>
<td>SCT0_OUT4</td>
<td>Apply glitch on SCL line or SDA line</td>
</tr>
</tbody>
</table>

5 \(\text{I}^2\text{C}\) transmission error type

5.1 Start/Stop error

According to \(\text{I}^2\text{C}\)-bus specification and user manual (document UM10204), all transactions begin with a START (S) and are terminated by a STOP (P). The bus is considered to be busy after a START condition. The bus is considered to be free again at a certain time after a STOP condition. In other words, if a Start or Stop condition is detected when bus is busy, it can be considered as a Start/Stop error.

5.2 Arbitration loss

According to \(\text{I}^2\text{C}\)-bus specification and user manual (document UM10204), arbitration which is only applicable to multi-master ensures that only one master is allowed to control bus and the data transfer is not corrupted, if more than one master simultaneously tries to control the bus. After the Start condition being issued, arbitration is then required to determine which master will complete its transmission.

The arbitration is performed bit by bit. For a specific bit, each bus master must check whether the bit data to be sent matches the current SDA line status or not. If they match, continue to compare the next bit until the last bit. Otherwise, the master will lose control of the bus, that is, an Arbitration Loss.

5.3 Time-out

LPC51U68 \(\text{I}^2\text{C}\) supports time-out feature and supports two timeout types, SCL time-out and Event time-out.

SCL time-out is reflected by the SCLTIMEOUT flag in the STAT register and is asserted when the SCL signal remains low longer than the time configured in the TIMEOUT register.
Event time-out is reflected by the EVENTTIMEOUT flag in the STAT register, the time between bus events governs the time-out check. These events include Start, Stop, and all changes on the I2C clock (SCL). This time-out is asserted when the time between any of these events is longer than the time configured in the TIMEOUT register.

6 Glitch generation for I2C transmission error

SCTimer is used to implement glitch generator. As the peripheral is only available in NXP MCUs, the state configurable timer (SCTimer/PWM) is available on all LPC55S69 devices. It can work like traditional timer as a timer/counter. However, the state machine function can be used to detect serial clock on SCL line, which is the most outstanding feature of SCTimer/PWM and greatly enhances the configuration and control flexibility of LPC devices.

6.1 Glitch generation for Start/Stop error

According to the description on Start/Stop error in I2C transmission error type, the bus is busy after the master issues a Start condition. Therefore, a Start or Stop condition is generated when the bus is busy is regarded as a Start/Stop error.

Figure 8 shows the state transition diagram for Start/Stop error. State 3 has checked the second rising edge of the I2C serial clock and then delays for a period of time, during the SCL high level period, pull down SDA. A Start condition is generated when I2C bus is busy since the master is addressing slave devices. Therefore, a Start/Stop error occurs.

Figure 8. State transition for Start/Stop error

Figure 9 shows the waveforms captured from SCL line, SDA line and glitch generator output. Due to the output of the glitch generator, a Start condition is generated during the high level of the second I2C serial clock, which is consistent with the setting of the state machine.

Figure 9 shows the waveforms captured from SCL line, SDA line and glitch generator output. Due to the output of the glitch generator, a Start condition is generated during the high level of the second I2C serial clock, which is consistent with the setting of the state machine.
6.2 Glitch generation for arbitration loss

According to the description on Arbitration Loss in I2C transmission error type, in the case of multi-master, when certain data bit to be sent by the master is inconsistent with the state of the SDA bus, the master loses control of the bus. At this time, an Arbitration Loss occurs.

Figure 10 shows the state transition diagram for Arbitration Loss. **State 2** has checked the second falling edge of the I2C serial clock and then delays for a period of time, during the SCL low level period, pull down SDA. The slave address is configured as 0x7E and is sent in Most Significant Bit (MSB) mode. In other words, the master wants to send a logic high to SDA during the second serial clock. However, the current state of SDA is logic low. Therefore, an Arbitration Loss occurs.
Figure 11 shows the waveforms captured from SCL line, SDA line and glitch generator output. Due to the output of the glitch generator, a mismatch between address bit status which master wants to send and current state of SDA line is generated during the low level of the second \( ^2\text{C} \) serial clock, which is consistent with the setting of the state machine.

![Waveforms for arbitration loss](image1)

**Figure 11. Waveforms for arbitration loss**

### 6.3 Glitch generation for SCL time-out

According to the description on SCL time-out in \( ^2\text{C} \) transmission error type, when the SCL signal remains low level longer than the time configured in the TIMEOUT register, an SCL time-out occurs.

Figure 12 shows the state transition diagram for SCL time-out. **State 4** has checked the third falling edge of the \( ^2\text{C} \) serial clock and pull down SCL. Since SCL will always be driven low and the duration of low level will be longer than the time configured in the TIMEOUT register, an SCL time-out occurs.

![State transition for SCL time-out](image2)

**Figure 12. State transition for SCL time-out**

Figure 13 shows the waveforms captured from SCL line, SDA line and glitch generator output. Due to the output of the glitch generator, SCL is kept at low level, which is consistent with the setting of the state machine.
6.4 Glitch generation for event time-out

According to the description on Event time-out in I2C transmission error type, when the time interval between any of bus events is longer than the time configured in the TIMEOUT register, an Event time-out occurs.

Figure 14 shows the state transition diagram for Event time-out. **State 0** drives SDA to low level when has checked the first falling edge of the I2C serial clock and then delays for a period of time, drives SDA to high level during SCL high level. This is a Stop condition.
Figure 15 shows the waveforms captured from SCL line, SDA line and glitch generator output. Due to the output of the glitch generator, the time interval between two adjacent Start/Stop events is 10.68 μs. It is longer than the time with the value of 10.67 μs configured in the TIMEOUT register, as described in Time-out threshold calculation.

Error Detect line in Figure 15 is used to indicate the detection to Start/Stop error and Event time-out. This line is initialized to low level. One rising edge appears on this line when Start/Stop error occurs and one falling edge appears on this line when Event time-out occurs.
7 I2C transmission error detection

7.1 SDK support

The SDK version used in this document is 2.8.2. The SDK with this version doesn't support I2C transmission error detection. To detect I2C transmission errors, follow the steps below and all modifications are specific to I2C master.

1. Download LPC51U68 SDK version 2.8.2.
2. Open I2C master demo project located in `/boards/lpcxpresso51u68/driver_examples/i2c/interrupt_b2b_transfer/master`.
3. By default, the I2C timeout function is disabled and needs to be enabled by software. In addition, set the timeout threshold of the TIMEOUT register to the minimum.
4. The macro `kStatus_I2C_Timeout` in the SDK can’t distinguish whether it is SCL time-out or Event time-out. For this reason, two macros are added to distinguish time-out types, as shown in the red square box in Figure 17.

5. By default, the interrupt flag of the I2C master in the SDK does not include SCL time-out and Event time-out. It is necessary to add supports for these two time-out types in the red square box.

7. This step is very important for detecting I2C Event time-out. If this step is missing, Event time-out can’t be detected. The reason for this step is described in Event time-out detection.
8. Add I^2C error output function to I^2C interrupt callback function marked as `i2c_master_callback`.

```c
void I2C_MasterTransferHandleIRQ(I2C_Type *base, I2C_master_handle_t *handle)
{
    bool isDone;
    status_t result;

    /* Don’t do anything if we don’t have a valid handle. */
    if (NULL == handle)
    {
        return;
    }

    result = I2C_RunTransferStateMachine(base, handle, aisDone);
    if ((result != kStatus_Success) || isDone)
    {
        /* Restore handle to idle state. */
        handle->state = (uint8_t)kIdleState;

        /* Disable internal IRQ enables. */
        I2C_DisableInterrupts(base, (uint32_t)kI2C_MasterIrqFlags);

        if (callback != NULL)
        {
            handle->completionCallback(base, handle, result, handle->userData);
        }
    }
    else
    {
        /* Enable internal IRQ enables. */
        I2C_EnableInterrupts(base, (uint32_t)kI2C_MasterIrqFlags);
    }
}
```

Figure 20. Modifications to detect event timeout

```c
static void i2c_master_callback(I2C_Type *base,
                                I2C_master_handle_t *handle,
                                status_t status,
                                void *userData)
{

    /* Signal transfer success when received success status. */
    if (status == kStatus_Success)
    {
        //IIC transfer success
        g_MasterCompletionFlag = true;
    }
    else
    {
        //IIC transfer generates error
        //output IIC error type
        showErrorInfo(status);
    }
}
```

Figure 21. Add I^2C error display function
7.2 Time-out threshold calculation

The I2C timeout threshold is determined by timeout register marked as TIMEOUT. Figure 23 shows the descriptions of the TIMEOUT register bit field and I2C function clock.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Symbol</th>
<th>Description</th>
<th>Reset value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3:0</td>
<td>TOMIN</td>
<td>Time-out time value, bottom four bits. These are hard-wired to 0xF. This gives a minimum time-out of 16 PC function clocks and also a time-out resolution of 16 I2C function clocks.</td>
<td>0xF</td>
</tr>
<tr>
<td>15:4</td>
<td>TO</td>
<td>Time-out time value. Specifies the time-out interval value in increments of 1 I2C function clocks. As defined by the CLKDIV register. To change this value while I2C is in operation, disable all time-outs, write a new value to TIMEOUT, then re-enable time-outs. 0x000 = A time-out will occur after 16 counts of the I2C function clock. 0x001 = A time-out will occur after 32 counts of the I2C function clock. 0xFFFF = A time-out will occur after 65,536 counts of the I2C function clock.</td>
<td>0xFFFF</td>
</tr>
<tr>
<td>31:16</td>
<td>-</td>
<td>Reserved. Read value is undefined, only zero should be written.</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 23. TIMEOUT register

Figure 24 shows the bit field arrangement of the CLKDIV register.
Figure 24. CLKDIV register

As shown in Figure 16, the bit field [15:4] in the TIMEOUT register is set to 0.

As shown in Figure 23, the timeout threshold is 16 \( \text{I}^\text{2}\text{C} \) function clocks, and the \( \text{I}^\text{2}\text{C} \) function clock is determined by the value of the CLKDIV register and the \( \text{I}^\text{2}\text{C} \) module clock.

As shown in Figure 25, the \( \text{I}^\text{2}\text{C} \) module clock frequency is set to 12 MHz in this document, and CLKDIV is 7.

As described above, the timeout threshold can be calculated as follows:

- \( \text{I}^\text{2}\text{C} \) clock divider value is \( \text{CLKDIV} + 1 = 8 \)
- \( \text{I}^\text{2}\text{C} \) function clock is \( 12\text{MHz} / 8 = 1.5\text{MHz} \)
- \( \text{I}^\text{2}\text{C} \) timeout threshold is \( 16 \times (1/(1.5\text{MHz})) = 16 \times ((2/3)\text{μs}) = 10.67\text{μs} \)

7.3 Event time-out detection

As described in \( \text{I}^\text{2}\text{C} \) transmission error type. Event time-out is asserted when the time interval between any of bus events is longer than the time configured in the TIMEOUT register. Bus events include Start, Stop, and all changes on the \( \text{I}^\text{2}\text{C} \) clock (SCL).

A key point is that Event time-out may be accompanied by Start/Stop error, and Start/Stop error precedes Event time-out. Therefore, \( \text{I}^\text{2}\text{C} \) interrupt can’t be disabled after Start/Stop error is detected, otherwise Event time-out will not be detected. This is why the \( \text{I}^\text{2}\text{C} \) interrupt is enabled in Step 7 in SDK support.

7.4 Detection output

This document supports two ways to observe detection output of \( \text{I}^\text{2}\text{C} \) transmission error:

- To print the \( \text{I}^\text{2}\text{C} \) transmission error type to the console on the PC side through UART
- To light up the LEDs connected to different IO pins when different types of \( \text{I}^\text{2}\text{C} \) transmission errors occur

Table 1 describes the correspondence between \( \text{I}^\text{2}\text{C} \) transmission error type and IO pin.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
\textbf{Bit} & \textbf{Description} \\
\hline
15:0 & DIVVAL  \\
\hline
\end{tabular}
\end{table}

Figure 25 shows the normal transmission between \( \text{I}^\text{2}\text{C} \) master and \( \text{I}^\text{2}\text{C} \) slave. Master sends data bytes from \( 0x00 \) to \( 0x1f \) and receives data bytes which are the same as the data bytes sent to slave.

Figure 26, Figure 27, Figure 28, and Figure 29 show the detection outputs for Start/Stop error, Arbitration Loss, SCL time-out and Event time-out.
Figure 25. Normal I2C transmission

Figure 26. Detection output for Start/Stop error
8 How to use LPC51U68 I2C glitch generation and detection demo

To use this demo, perform the following steps:

1. Establish hardware environment for this demo as described in Hardware configurations.

2. Open I2C master code project located in I2C_Transfer_With_Error_Detection\boards\lpcxpresso51u68\driver_examples\i2c\interrupt_b2b_transfer\master\mdk and compile it to generate an executable file.
3. Open \textit{I^2C} slave code project located in 
\texttt{I2C\_Transfer\_With\_Error\_Detection\boards\lpcxpresso51u68\driver\examples\i2c\interrupt\b2b\_transfer\slave\mdk} and compile it to generate an executable file.

4. Open glitch generator code project located in 
\texttt{I2C\_Glitch\_Generator\boards\lpcxpresso55s69\driver\examples\sctimer\edge\check\match\cm33\core\0\mdk} and compile it to generate an executable file.

5. Program executable file for \textit{I^2C} master, \textit{I^2C} slave and glitch generator to corresponding evaluation boards mentioned in \textbf{System overview}.

6. Open two UART terminals on PC side corresponding to master and slave. The settings are as below:
   - Baudrate:115200
   - Data bits:8
   - Parity Check: No
   - Flow Control: No

7. Run the following in sequence:
   a. Glitch generator application
   b. \textit{I^2C} slave application
   c. \textit{I^2C} master application

9 \textbf{Summary}

This document introduces:
   - \textit{I^2C} bus
   - composition of the system
   - role of each component
   - primary transmission errors of \textit{I^2C} bus including Start/Stop error, Arbitration Loss, SCL time-out and Event time-out
   - how to design glitch generator using SCTimer state machine
   - how to detect \textit{I^2C} transmission errors triggered by glitch
   - how to use generation and detection demo for the \textit{I^2C} transmission error

Demo code projects are attached to this document. For more technical details, see AN13238SW.
Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

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.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. 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.