Telit MT3333 I2C Application Note

80434NT11630A Rev. 0
2017-11-17

TELIT
TECHNICAL
DOCUMENTATION

SPECIFICATIONS ARE SUBJECT TO CHANGE WITHOUT NOTICE

NOTICES .

While reasonable efforts have been made to assure the accuracy of this document, Telit assumes no liability resulting from any inaccuracies or omissions in this document, or from use of the information obtained herein. The information in this document has been carefully checked and is believed to be reliable. However, no responsibility is assumed for inaccuracies or omissions. Telit reserves the right to make changes to any products described herein and reserves the right to revise this document and to make changes from time to time in content hereof with no obligation to notify any person of revisions or changes. Telit does not assume any liability arising out of the application or use of any product, software, or circuit described herein; neither does it convey license under its patent rights or the rights of others.
It is possible that this publication may contain references to, or information about Telit products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that Telit intends to announce such Telit products, programming, or services in your country.

COPYRIGHTS

This instruction manual and the Telit products described in this instruction manual may be, include or describe copyrighted Telit material, such as computer programs stored in semiconductor memories or other media. Laws in the Italy and other countries preserve for Telit and its licensors certain exclusive rights for copyrighted material, including the exclusive right to copy, reproduce in any form, distribute and make derivative works of the copyrighted material. Accordingly, any copyrighted material of Telit and its licensors contained herein or in the Telit products described in this instruction manual may not be copied, reproduced, distributed, merged or modified in any manner without the express written permission of Telit. Furthermore, the purchase of Telit products shall not be deemed to grant either directly or by implication, estoppel, or otherwise, any license under the copyrights, patents or patent applications of Telit, as arises by operation of law in the sale of a product.

COMPUTER SOFTWARE COPYRIGHTS

The Telit and 3rd Party supplied Software (SW) products described in this instruction manual may include copyrighted Telit and other 3rd Party supplied computer programs stored in semiconductor memories or other media. Laws in the Italy and other countries preserve for Telit and other 3rd Party supplied SW certain exclusive rights for copyrighted computer programs, including the exclusive right to copy or reproduce in any form the copyrighted computer program. Accordingly, any copyrighted Telit or other 3rd Party supplied SW computer programs contained in the Telit products described in this instruction manual may not be copied (reverse engineered) or reproduced in any manner without the express written permission of Telit or the 3rd Party SW supplier. Furthermore, the purchase of Telit products shall not be deemed to grant either directly or by implication, estoppel, or otherwise, any license under the copyrights, patents or patent applications of Telit or other 3rd Party supplied SW, except for the normal non-exclusive, royalty free license to use that arises by operation of law in the sale of a product.

USAGE AND DISCLOSURE RESTRICTIONS

I. License Agreements

The software described in this document is the property of Telit and its licensors. It is furnished by express license agreement only and may be used only in accordance with the terms of such an agreement.

II. Copyrighted Materials

Software and documentation are copyrighted materials. Making unauthorized copies is prohibited by law. No part of the software or documentation may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, without prior written permission of Telit

III. High Risk Materials

Components, units, or third-party products used in the product described herein are NOT fault-tolerant and are NOT designed, manufactured, or intended for use as on-line control equipment in the following hazardous environments requiring fail-safe controls: the operation of Nuclear Facilities, Aircraft Navigation or Aircraft Communication Systems, Air Traffic Control, Life Support, or Weapons Systems (High Risk Activities"). Telit and its supplier(s) specifically disclaim any expressed or implied warranty of fitness for such High Risk Activities.

IV. Trademarks

TELIT and the Stylized T Logo are registered in Trademark Office. All other product or service names are the property of their respective owners.

V. Third Party Rights

The software may include Third Party Right software. In this case you agree to comply with all terms and conditions imposed on you in respect of such separate software. In addition to Third Party Terms, the disclaimer of warranty and limitation of liability provisions in this License shall apply to the Third Party Right software.
TELIT HEREBY DISCLAIMS ANY AND ALL WARRANTIES EXPRESS OR IMPLIED FROM ANY THIRD PARTIES REGARDING ANY SEPARATE FILES, ANY THIRD PARTY MATERIALS INCLUDED IN THE SOFTWARE, ANY THIRD PARTY MATERIALS FROM WHICH THE SOFTWARE IS DERIVED (COLLECTIVELY “OTHER CODE”), AND THE USE OF ANY OR ALL THE OTHER CODE IN CONNECTION WITH THE SOFTWARE, INCLUDING (WITHOUT LIMITATION) ANY WARRANTIES OF SATISFACTORY QUALITY OR FITNESS FOR A PARTICULAR PURPOSE.
NO THIRD PARTY LICENSORS OF OTHER CODE SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND WHETHER MADE UNDER CONTRACT, TORT OR OTHER LEGAL THEORY, ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE OTHER CODE OR THE EXERCISE OF ANY RIGHTS GRANTED UNDER EITHER OR BOTH THIS LICENSE AND THE LEGAL TERMS APPLICABLE TO ANY SEPARATE FILES, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

1. PRODUCT APPLICABILITY TABLE

PRODUCT	DRI SUPPORTED
SE868K3-A	NO
SL869L-V2	NO
SL871	NO

Table 1-1 Product Applicability

2. CONTENTS

3. INTRODUCTION

3.1. Scope

This document describes the process of reading and writing data from the I^2C bus of modules based on the MTK3333 chip. It includes example flow process, and C language example implementation.

3.2. Audience

This document is intended for developers who are writing or editing software for Telit GNSS modulles based on the MT3333.

3.3. Contact Information, Support

For general contact, technical support services, technical questions and report documentation errors contact Telit Technical Support at:

- TS-EMEA@telit.com
- TS-AMERICAS@telit.com
- TS-APAC@telit.com

Alternatively, use:
http://www.telit.com/support

For detailed information about where you can buy the Telit modules or for recommendations on accessories and components visit:
http://www.telit.com

Our aim is to make this guide as helpful as possible. Keep us informed of your comments and suggestions for improvements.
Telit appreciates feedback from the users of our information.

3.4. Text Conventions

Danger – This information MUST be followed or catastrophic equipment failure or bodily injury may occur.

Caution or Warning – Alerts the user to important points about integrating the module, if these points are not followed, the module and end user equipment may fail or malfunction.

Tip or Information – Provides advice and suggestions that may be useful when integrating the module.

All dates are in ISO 8601 format, i.e. YYYY-MM-DD.

3.5. Related Documents

Documents and Downloads are afailable at:
https://www.telit.com/products/positioning-and-timing/

- SL871 Datasheet
- SL871 Product User Guide
- SL871 EVK User Guide

4. I²C INTERFACE

This document describes the process of reading and writing data from the Inter Integrated Circuit (I²C) bus.
It includes example flow process, and C language example implementation.

In this document, the Master refers to the Host Device or Processor, and Slave refers to the module.

Supported Features:

- Fast mode, bit rete up to 400Kbit/s
- 7-bit address.
- Slave mode only.
- Default Slave address is 0x10

Note: MT3333 does not support firmware upgrade through I²C.

Note: Interrupt Mode is not supported on all modules. Contact Telit technical support for further information.

5. I²C READ OPERATION

5.1. I²C Receive Data Flow

5.1.1. General I²C Data Format

The below figure shows the general data I²C packet structure, which complies with the I²C standard.

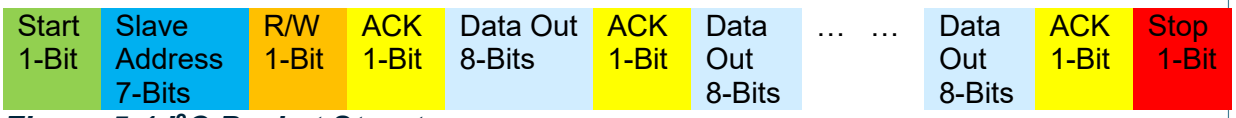


Figure 5-1 I²C Packet Structure

5.1.2. I²C Buffer

The I2C buffer is 255 bytes long. The Master can read a maximum packet size of 255 bytes at a time.
The Master must be capable of reading several I²C data packets in one second, and extracting the NMEA data from them.

5.1.3. Timing Requirements

The module requires 5ms to upload new data into buffer. After reading the entire 255 bytes of data out of the buffer, the master must wait 5ms before reading the next set of data.
If the entire one second of NMEA packets are read, the Master can extend the wait period before the entire NMEA packet of next second is ready.

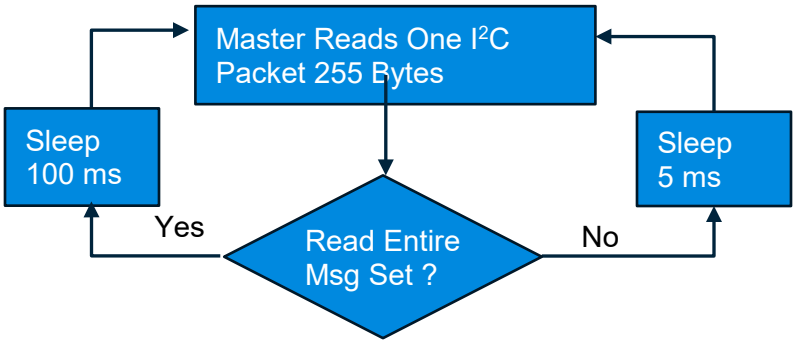
5.2. Reading modes for the I²C port

There are two read modes for I²C ports: Polling mode and Interupt mode as described below.

5.2.1. Polling Mode

Data is kept in the buffer until it is read out by the Master. The Master must read the entire one second of NMEA packets in a polling time interval to avoid data loss.

When the I²C buffer is full, no new data will be loaded. If the data is not read out, new data from the module will be lost.
This time interval should be configured according to GNSS fix interval, and should be less than GNSS fix interval to guarantee that no data will be lost.



Note: The above figure assume GNSS fix interval is 1 second, so set polling time interval to 100ms.

Figure 6-2 Polling Mode Master Reading Flow

5.2.2. Interrupt Mode

Note: Interrupt Mode is not supported on all modules. Contact Telit technical support for further information.

When there is data in the I²C buffer, the interrupt pin will be high. When the entire buffer is read and empty, the interrupt pin be low.

Note: Polling mode can also use the interrupt pin to determine if there is data in the I²C buffer.

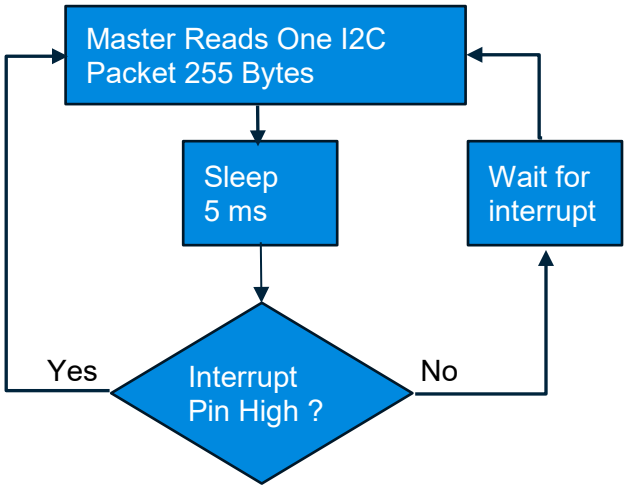


Figure 6-3 Interrupt Mode Master Reading Flow

5.2.3. Reading Mode Comparison

Comparing polling mode versus Interrupt mode, interrupt mode will receive less unusable bytes in the I²C data packets, and saves the Master from unnecessary processing. However, an available external interrupt GPIO is required on the Master device.

5.3. I²C Data Packet Format

NMEA data, less than or equal to 254 bytes	0x0A End Char 1 Byte

Figure 5-4 NMEA Data Packet Format

5.3.1. Reading the Buffer

The I²C data buffer is 255 bytes long, with up to 254 data bytes and one End character, Line Feed <LF> (0x0A).
The Master can read 255 bytes maximum at a time from the I²C buffer. When the buffer is empty, the master will read unused bytes in the data packet of 255 bytes.
Example (full buffer):
There are 254 NMEA data bytes and 1 end Char <LF> (0x0A) in the buffer as shown in the figure below.

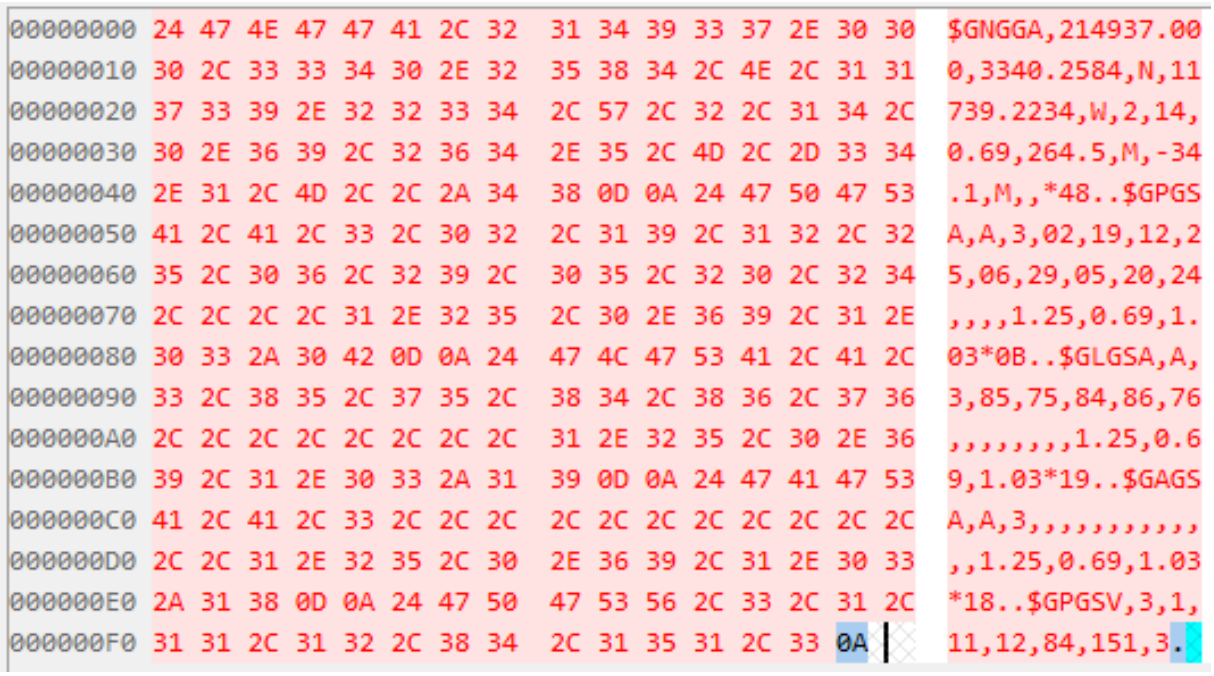


Figure 6-5 Data Read from the full I²C Buffer

5.3.2. I²C Master Read Format

There are three cases the master will encounter when reading data from the I²C port.

5.3.2.1. Buffer Empty

The case where I²C buffer is empty, and the 255 bytes contains no data, only unused bytes (0x0A) as shown in the fugure below.

Example:

Figure 6-6 Data Read from the empty I²C Buffer with all Unused bytes

5.3.2.2. Buffer Partially Full

The case where I²C buffer already has existing data, and the 255 bytes contains NMEA data at the beginning of the packet and some unused bytes (0x0A) at the end of thepacket.

Example:
If the buffer has 206 bytes of NMEA data, and the Master reads it (255 bytes), it will be read in as follows, which include unused bytes.

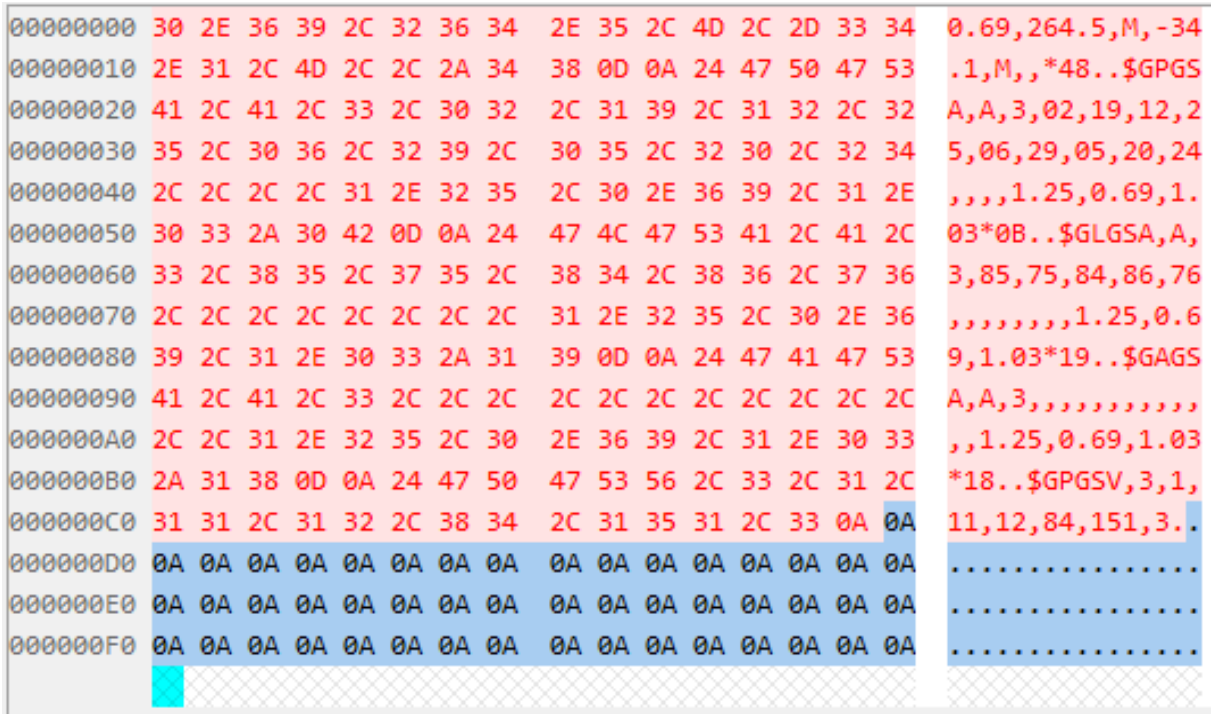


Figure 5-7 Data Read from the I²C Buffer with some Unused bytes

Note: The Unused bytes are ‘0A’, because when the buffer is empty, it will output last valid byte repeatedly until new data upload into the buffer. The last valid byte in a NMEA sentence is ‘0x0A’.

5.3.2.3. Buffer Filling During Read

The case where the I²C buffer is initially empty, and during the process of the Master reading, the module loads data into it.

Example:

```
00000000 0A 0A 0A 0A 0A 0A 0A 0A  0A 0A 0A 0A 0A 0A 0A 0A  ................
00000010 0A 0A 0A 0A 0A 0A 0A 0A  0A 0A 0A 0A 0A 0A 0A 0A  ................
00000020 0A 0A 0A 0A 0A 0A 0A 0A  0A 0A 0A 0A 0A 0A 0A 0A  ................
00000030 0A 0A 0A 0A 0A 0A 0A 0A  0A 0A 0A 0A 0A 0A 0A 0A  ................
00000040 0A 0A 0A 0A 0A 0A 0A 0A  0A 0A 0A 0A 0A 0A 0A 0A  ................
00000050 0A 0A 24 47 4E 47 47 41  2C 32 32 32 31 34 39 2E  ..$GNGGA,222149.
00000060 30 30 30 2C 33 33 34 30  2E 32 35 38 32 2C 4E 2C  000,3340.2582,N,
00000070 31 31 37 33 39 2E 32 32  33 34 2C 57 2C 32 2C 31  11739.2234,W,2,1
00000080 36 2C 30 2E 36 33 2C 32  36 33 2E 36 2C 4D 2C 2D  6,0.63,263.6,M,-
00000090 33 34 2E 31 2C 4D 2C 2C  2A 34 36 0D 0A 24 47 50  34.1,M,,*46..$GP
000000A0 47 53 41 2C 41 2C 33 2C  30 32 2C 30 36 2C 31 32  GSA,A,3,02,06,12
000000B0 2C 32 35 2C 32 31 2C 32  39 2C 30 35 2C 32 30 2C  ,25,21,29,05,20,
000000C0 32 34 2C 33 31 2C 2C 2C  31 2E 30 37 2C 30 2E 36  24,31,,,1.07,0.6
000000D0 33 2C 30 2E 38 36 2A 30  34 0D 0A 24 47 4C 47 53  3,0.86*04..$GLGS
000000E0 41 2C 41 2C 33 2C 38 35  2C 37 31 2C 37 35 2C 38  A,A,3,85,71,75,8
000000F0 34 2C 38 36 2C 37 36 2C  2C 2C 2C 2C 2C 2C 31 2E  4,86,76,,,,,,,1.
```

Figure 6-8 Data Read from the I²C Buffer with Partial Unused Bytes

5.3.3. Extracting NMEA data from several I²C packets

As the above section described, NMEA data for one reporting period may need to be extracted from several I²C packets. Sample code is provided for extracting valid NMEA data. It will be introduced in Section **7.1.1 Read Example.**

Note: When extracting NMEA data from several I²C packets, all '0A' characters should be discarded. This includes the following three situations:

1. The end char of one I²C packet '0A'
2. Unused bytes ('0A' under normal circumstances)
3. The <LF> char '0A' terminating each NMEA sentence

6. I²C WRITE OPERATION

The user can input NMEA commands to the I²C port. The module I²C RX buffer has 255 bytes, so one I²C packet that master sends should be less than 255 bytes. The minimum time interval between two input I²C packets is 30 milliseconds because module needs that amount of time to process input data.

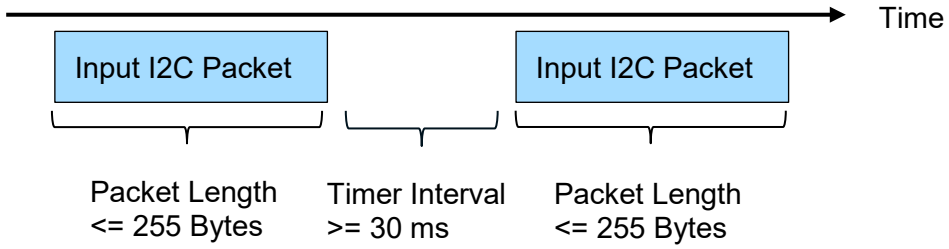


Figure 7-1 I²C Write Timing

7. EXAMPLE CODE

These examples are intended to demonstrate low level controlling of the Host/Master GPIO to read and write data from the I²C interface.

7.1.1. Read Example

The following is an example of implementing reading from the module by a Master.

```
#define SDA 0xXX  // User must add an I/O port for SDA
#define SCL 0xXX  // User must add an I/O port for SCL

/*************************************************************
 * i2cRead
 * Reads an 8-bit register with the I2C port.
 * Data is returned.
 *************************************************************/

unsigned char i2cRead (const unsigned char regAddr)
{
  unsigned char i2cCount;           // Counter used to clock out the data
  unsigned char i2cData;

  i2cData = regAddr;    // Preload the data to be sent with Address and Data

  SDA = 0;                          // Pull SDA low to indicate start bit
  for (i2cCount = 0; i2cCount < 8; i2cCount++)    // Prepare to clock out
 {                                                // the Address byte
    if (i2cData & 0x80)
           SDA = 1;
    else
           SDA = 0;
       SCL = 0;
       SCL = 1;
    i2cData <<= 1;
  }                                 // and loop back to send the next bit

       SDA = 1;                     // Reset the data line

  i2cData = 0;
  for (i2cCount = 0; i2cCount < 8; i2cCount++)   // Prepare to clock in the
  {                                              // data to be read
    i2cData <<=1;                   // Rotate the data
    SCL = 0;                        // Drop the clock to clock the data out
    i2cData += SDA;                 // Read the data bit
    SCL = 1;                        // Raise the clock ready for the next bit
  }                                 // and loop back

  return ((unsigned char)i2cData); // Return the read data
```

7.1.2. Write Example

The following is an example of implementing writing to the module by a Master.

```
#define SDA 0xXX  // User must add an I/O port for SDA
#define SCL 0xXX  // User must add an I/O port for SCL

/************************************************************
* i2cWrite
* Writes to an 8-bit register with the i2c port
************************************************************/

void i2cWrite(const unsigned char regAddr, const unsigned char regData)
{
  unsigned char i2cCount;     // Counter used to clock out the data
  unsigned char i2cData;


  i2cData = regAddr;          // Preload the data to be sent with Address
      SDA = 0;                // Pull SDA low to indicate start bit

  for (i2cCount = 0; i2cCount < 8; i2cCount++) // Prepare to clock out the
   {                                           // Address byte
    if (i2cData & 0x80)        // Check for a 1
      SDA = 1;                 // and set the SDA line appropriately
    else
      SDA = 0;
    SCL = 1;                   // Toggle the clock line
    SCL = 0;
    i2cData <<= 1;             // Rotate to get the next bit
  }                            // and loop back to send the next bit

  // Repeat for the Data byte

  i2cData = regData;           // Preload the data to be sent with Data
  for (i2cCount = 0; i2cCount < 8; i2cCount++)
  {
    if (i2cData & 0x80)
      SDA = 1;
    else
      SDA = 0;
    SCL = 1;
    SCL = 0;
    i2cData <<= 1;
  }

  SDA = 1;
  SCL = 1;
}
```

7.2. Example NMEA Read

This example extracts NMEA data from the module and discards the unused bytes.

7.2.1. Functions and Flow

Function	Description
iop_init_pcrx	Initialize receive queue.
iop_inst_avail	Get available NMEA sentence information.
iop_get_inst	Get NMEA sentence data from queue buffer.
iop_pcrx_nmea	Process incoming data, get valid NMEA data, and discard unused bytes.

Figure 7-1 Functions

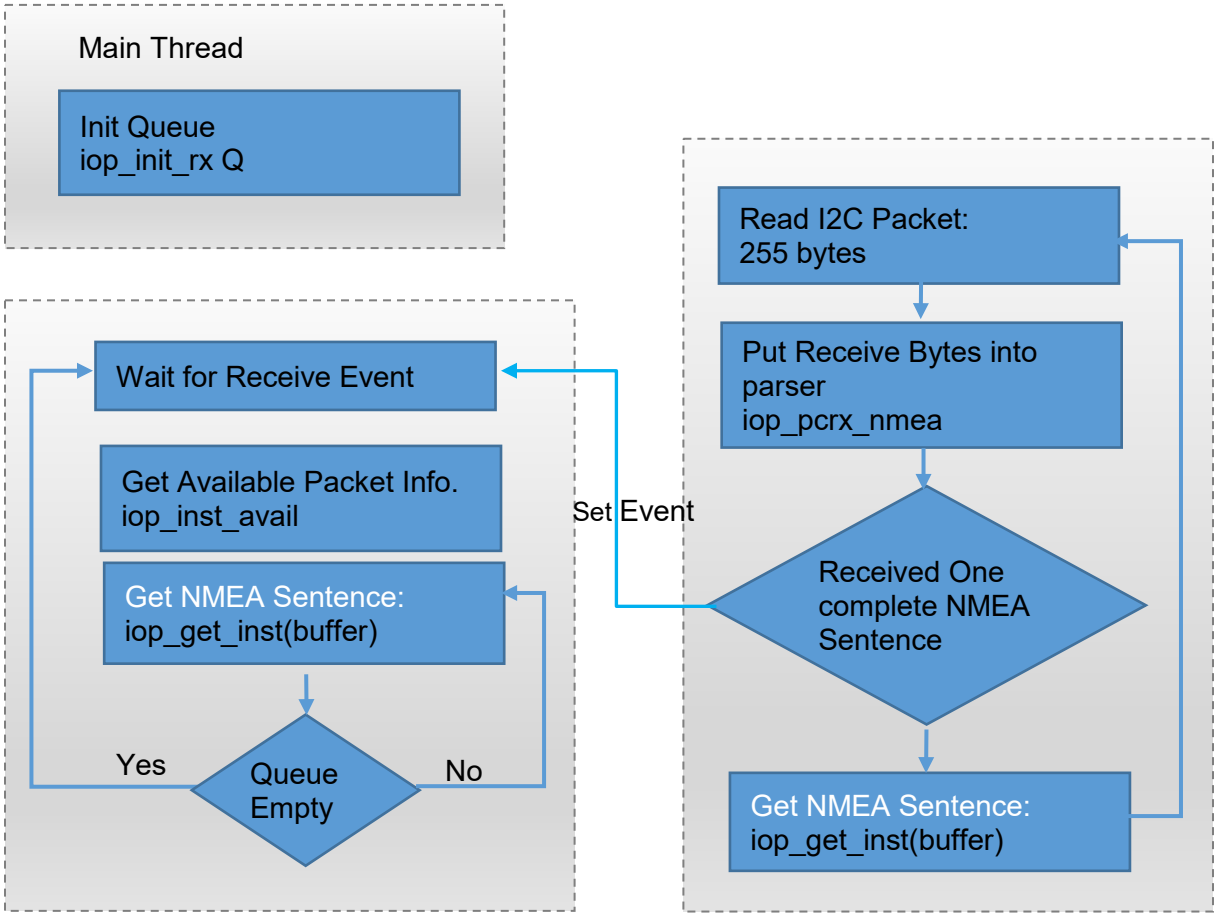


Figure 8-2 NMEA processing flow

7.2.2. NMEA Read Implementation Example

```
/* ================================================================== */
/* Defines                                                            */
/* ================================================================== */
#define IOP_LF_DATA 0x0A
#define IOP_CR_DATA 0x0D
#define IOP_START_NMEA 0x24
#define NMEA_ID_QUE_SIZE 0x0100 // Configure according to resource condition
#define NMEA_RX_QUE_SIZE 0x8000 // Configure according to resource condition
#define MAX_I2C_PKT_LEN 255
#define MAX_NMEA_STN_LEN 256

typedef enum
{
  RXS_DAT,                    // receive NMEA data
  RXS_ETX,                    // End-of-packet
} RX_SYNC_STATE_T;

struct
{
    short   inst_id;          // 1 - NMEA, 2 - DBG, 3 - HBD
    short   dat_idx;
    short   dat_siz;
} id_que[NMEA_ID_QUE_SIZE];

RX_SYNC_STATE_T rx_state;

char rx_que[NMEA_RX_QUE_SIZE];
unsigned short id_que_head;
unsigned short id_que_tail;
unsigned short rx_que_head;

unsigned int u4SyncPkt;
unsigned int u4OverflowPkt;
unsigned int u4PktInQueue;

/* ================================================================== */
/* Functions                                                          */
/* ================================================================== */
//Queue Functions
BOOL iop_init_pcrx( void )
{
    /*-------------------------------------------------------------
    variables
    -------------------------------------------------------------*/
    short   i;

    /*-------------------------------------------------------------
    initialize queue indexes
    -------------------------------------------------------------*/
    id_que_head = 0;
    id_que_tail = 0;
    rx_que_head = 0;
```

```
    /*-------------------------------------------------------
    initialize identification queue
    -------------------------------------------------------*/
    for( i=0; i< NMEA_ID_QUE_SIZE; i++)
    {
        id_que[i].inst_id = -1;
        id_que[i].dat_idx =  0;
    }

    /*-------------------------------------------------------
    initialize receive state
    -------------------------------------------------------*/
    rx_state = RXS_ETX;

    /*-------------------------------------------------------
    initialize statistic information
    -------------------------------------------------------*/
    u4SyncPkt = 0;
    u4OverflowPkt = 0;
    u4PktInQueue = 0;

    return TRUE;
}

/*****************************************************************
*   PROCEDURE NAME:
*       iop_inst_avail - Get available NMEA sentence information
*
*   DESCRIPTION:
*     inst_id - NMEA sentence type
*     dat_idx - start data index in queue
*     dat_siz - NMEA sentence size
*****************************************************************/
BOOL iop_inst_avail(short *inst_id, short *dat_idx, short *dat_siz)
{
    /*-------------------------------------------------------
    variables
    -------------------------------------------------------*/
    BOOL    inst_avail;

    /*-------------------------------------------------------
    if packet is available then return id and index
    -------------------------------------------------------*/
    if ( id_que_tail != id_que_head )
    {
        *inst_id = id_que[ id_que_tail ].inst_id;
        *dat_idx = id_que[ id_que_tail ].dat_idx;
        *dat_siz = id_que[ id_que_tail ].dat_siz;
        id_que[ id_que_tail ].inst_id = -1;
        id_que_tail = ++id_que_tail & (unsigned short)(NMEA_ID_QUE_SIZE - 1);
        inst_avail = TRUE;
        if (u4PktInQueue > 0)
        {
            u4PktInQueue--;
        }

    }
```

```
    else
    {
        inst_avail = FALSE;
    }
    return ( inst_avail );
}   /* iop_inst_avail() end */


/*************************************************************
*   PROCEDURE NAME:
*       iop_get_inst - Get available NMEA sentence from queue
*
*   DESCRIPTION:
*     idx - start data index in queue
*     size - NMEA sentence size
*     data - data buffer used to save NMEA sentence
*************************************************************/
void iop_get_inst(short idx, short size, void *data)
{
    /*-----------------------------------------------------------
    variables
    -----------------------------------------------------------*/
    short i;
    unsigned char *ptr;

    /*-----------------------------------------------------------
    copy data from the receive queue to the data buffer
    -----------------------------------------------------------*/
    ptr = (unsigned char *)data;
    for (i = 0; i < size; i++)
    {
        *ptr = rx_que[idx];
        ptr++;
        idx = ++idx & (unsigned short)(NMEA_RX_QUE_SIZE - 1);
    }
}   /* iop_get_inst() end */


/*************************************************************
*   PROCEDURE NAME:
*       iop_pcrx_nmea - Receive NMEA code
*
*   DESCRIPTION:
*     The procedure fetch the characters between/includes '$' and <CR>.
*     That is, character <CR><LF> is skipped.
*     and the maximum size of the sentence fetched by this procedure is 256
*     $xxxxxx*AA
*
*************************************************************/
void iop_pcrx_nmea( unsigned char in_data[],int i4NumByte )
{

int i;
unsigned char data;

    for (i = 0; i < i4NumByte; i++)
    {
```

```
        data = in_data[i];
        if (data == IOP_LF_DATA)
            {
                  continue;
        }

      /*-------------------------------------------------------
      determine the receive state
      -------------------------------------------------------*/
    switch (rx_state)
        {
        case RXS_DAT:
            switch (data)
                {
                case IOP_CR_DATA:
                    // count total number of sync packets
                    u4SyncPkt += 1;

                    id_que_head = ++id_que_head & (unsigned
                                          short)(NMEA_ID_QUE_SIZE - 1);
                    if (id_que_tail == id_que_head)
                    {
                      // count total number of overflow packets
                      u4OverflowPkt += 1;

                      id_que_tail = ++id_que_tail & (unsigned
                                          short)(NMEA_ID_QUE_SIZE - 1);
                    }
                    else
                    {
                      u4PktInQueue++;
                    }

                    rx_state = RXS_ETX;
                    /*-----------------------------------------
                    set RxEvent signaled
                    -----------------------------------------*/
                    SetEvent(hRxEvent);
                    break;

                case IOP_START_NMEA:
                {
                  // Restart NMEA sentence collection
                  rx_state = RXS_DAT;
                  id_que[id_que_head].inst_id = 1;
                  id_que[id_que_head].dat_idx = rx_que_head;
                  id_que[id_que_head].dat_siz = 0;
                  rx_que[rx_que_head] = data;
                  rx_que_head = ++rx_que_head & (unsigned
                                          short)(NMEA_RX_QUE_SIZE - 1);
                  id_que[id_que_head].dat_siz++;

                  break;
                }

                default:
                    rx_que[rx_que_head] = data;
```

```
                    rx_que_head = ++rx_que_head & (unsigned
                                                   short)(NMEA_RX_QUE_SIZE - 1);
                    id_que[id_que_head].dat_siz++;

          // if NMEA sentence length > 256, stop NMEA sentence collection
                    if (id_que[id_que_head].dat_siz == MAX_NMEA_STN_LEN)
                    {
                        id_que[id_que_head].inst_id = -1;

                        rx_state = RXS_ETX;
                    }
                    break;
                }
            break;

        case RXS_ETX:
            if (data == IOP_START_NMEA)
            {
                rx_state = RXS_DAT;
                id_que[id_que_head].inst_id = 1;
                id_que[id_que_head].dat_idx = rx_que_head;
                id_que[id_que_head].dat_siz = 0;
                rx_que[rx_que_head] = data;
                rx_que_head = ++rx_que_head & (unsigned
                                                short)(NMEA_RX_QUE_SIZE - 1);
                id_que[id_que_head].dat_siz++;
            }

            break;

        default:
            rx_state = RXS_ETX;

            break;
        }
      }
}   /* iop_pcrx_nmea() end */
```

8. DOCUMENT HISTORY

Revision	Date	Changes
0	2017-11-17	Initial Release
1		
2		

SUPPORT INQUIRIES

Link to **www.telit.com** and contact our technical support team for any questions related to technical issues.

www.telit.com

Telit Communications S.p.A.
Via Stazione di Prosecco, 5/B
I-34010 Sgonico (Trieste), Italy

Telit IoT Platforms LLC
5300 Broken Sound Blvd, Suite 150
Boca Raton, FL 33487, USA

Telit Wireless Solutions Inc.
3131 RDU Center Drive, Suite 135
Morrisville, NC 27560, USA

Telit Wireless Solutions Co., Ltd.
8th Fl., Shinyoung Securities Bld.
6, Gukjegeumyung-ro8-gil, Yeongdeungpo-gu
Seoul, 150-884, Korea

Telit Wireless Solutions Ltd.
10 Habarzel St.
Tel Aviv 69710, Israel

Telit Wireless Solutions
Technologia e Servicos Ltda
Avenida Paulista, 1776, Room 10.C
01310-921 São Paulo, Brazil

[01.2017]

Mod. 0809
2017-01