LE910Cx Linux Device Driver

Application Note

80502NT11769A Rev. 7 – 2022-03-14

Contents

APPLICABILITY TABLE

PRODUCTS
LE910C1-NA
LE910C1-NS
LE910C1-NF
LE910C4-NF
LE910C1-EU
LE910C4-EU
LE910C1-AP
LE910C4-AP
LE910C1-LA
LE910C4-LA
LE910C4-CN

1. INTRODUCTION

1.1. Scope

This document provides the descriptions and example code for controlling and configuring the interfaces.

1.2. Audience

This document is intended for Telit customers, especially system integrators, about to implement their applications using the Telit LE910Cx module.

1.3. Contact Information, Support

For technical support and general questions please e-mail:

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

Alternatively, use:

https://www.telit.com/contact-us/

Product information and technical documents are accessible 24/7 on our web site:

https://www.telit.com

1.4. Symbol Conventions

	Danger: This information MUST be followed, or catastrophic equipment failure or personal injury may occur.
	Warning: Alerts the user on important steps about the module integration.
	Note/Tip: Provides advice and suggestions that may be useful when integrating the module.
	Electro-static Discharge: Notifies the user to take proper grounding precautions before handling the product.

Table 1: Symbol Conventions

All dates are in ISO 8601 format, that is YYYY-MM-DD.

1.5. Related Documents

- LE910Cx AT Commands Reference Guide, 80502ST10950A
- LE910Cx Software User Guide, 1VV0301556
- LE910Cx Hardware User Guide, 1VV0301298

2. I2C INTERFACE

LE910Cx has a single I2C port and only supports master mode.

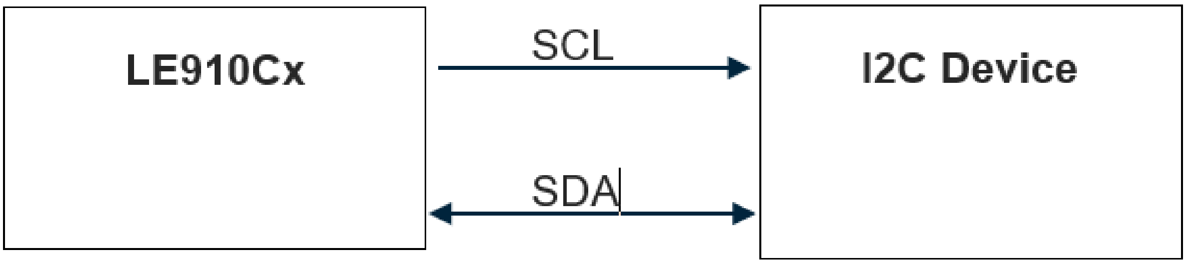

Figure 1: I2C Master Mode Interface

The following pins on the LE910Cx support an I2C interface:

- B11 - I2C_SCL
- B10 - I2C_SDA

2.1. Using I2C Interface

The I2C interface can be used externally by the end-user application. The I2C interface is accessible from the Linux driver device node(/dev/i2c-4).

Example:

```
#define I2C_4_DEV_NAME "/dev/i2c-4"

static int i2c_write(int fd,unsigned char slave_addr,unsigned char
reg,unsigned char value)
{
  unsigned char outbuf[2];
  struct i2c_rdwr_ioctl_data packets;
  struct i2c_msg messages[1];

  messages[0].addr  = slave_addr;
  messages[0].flags = 0;
  messages[0].len   = sizeof(outbuf);
  messages[0].buf   = outbuf;

  /* The first byte indicates which register we'll write */
  outbuf[0] = reg;

  /*
   * The second byte indicates the value to write.  Note that for many
```

```
     * devices, we can write multiple, sequential registers at once by
     * simply making outbuf bigger.
     */
    outbuf[1] = value;

    /* Transfer the i2c packets to the kernel and verify it worked */
    packets.msgs  = messages;
    packets.nmsgs = 1;
    if (ioctl(fd, I2C_RDWR, &packets) < 0) {
      perror("[I2C] Unable to send data");
      return 1;
    }
    return 0;
}

static int i2c_read(int file,unsigned char addr,unsigned char reg,unsigned
char *val)
{
    unsigned char inbuf, outbuf;
    struct i2c_rdwr_ioctl_data packets;
    struct i2c_msg messages[2];

    /*
     * In order to read a register, we first do a "dummy write" by writing
     * 0 bytes to the register we want to read from. This is similar to
     * the packet in set_i2c_register, except it's 1 byte rather than 2.
     */
    outbuf = reg;
    messages[0].addr  = addr;
    messages[0].flags = 0;
    messages[0].len   = sizeof(outbuf);
    messages[0].buf   = &outbuf;

    /* The data will get returned in this structure */
    messages[1].addr  = addr;
    messages[1].flags = I2C_M_RD/* | I2C_M_NOSTART*/;
    messages[1].len   = sizeof(inbuf);
    messages[1].buf   = &inbuf;

    /* Send the request to the kernel and get the result back */
    packets.msgs      = messages;
```

```
    packets.nmsgs     = 2;
    if(ioctl(file, I2C_RDWR, &packets) < 0) {
        perror("[I2C] Unable to send data");
        return 1;
    }
    *val = inbuf;

    return 0;
}

int main(int argc, char **argv)
{
  int i2c_fd = NULL;
  …
  // Open a connection to the I2C userspace control file.
  if ((i2c_fd = open(I2C_4_DEV_NAME, O_RDWR)) < 0) {
    perror("[I2C] Unable to open i2c_4 control file");
    exit(1);
}
i2c_write(i2c_fd,…….);
i2c_read(i2c_fd,…….);
close(i2c_fd);
return 0;
}
```

3. HSIC INTERFACE

LE910Cx provides a two-wire HSIC interface and supports HSIC master/ slave mode.

The LE910Cx HSIC interface supports the following features:

- No hot plug detection
- No hot removal/attachment, interface is always connected
- No high-speed chirp protocols
- HSIC master/slave mode support

3.1. HSIC Signaling

Table 2, details all the basic signaling protocols for HSIC. Many signals, such as CONNECT/RESUME and IDLE/SUSPEND are equivalent.

Bus State	Strobe	Data	Description
IDLE	High	Low	1 or more Strobe-periods
CONNECT	Low	High	2 Strobe-periods
RESUME	Low	High	For time periods per USB 2.0 specification
SUSPEND	High	Low	Identical to IDLE state
RESET	Low	Low	Per USB 2.0 specification

Table 2: HSIC Signaling Summary

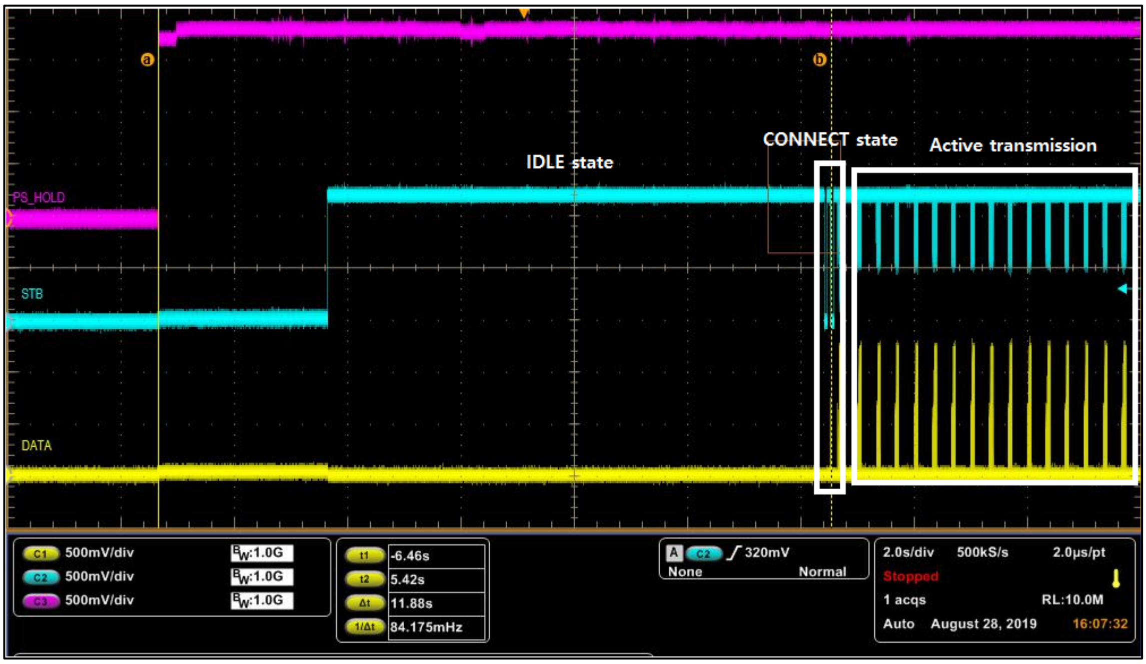

Figure 2: IDLE to CONNECT Signaling Example (LE910Cx Master and LE910Cx Slave)

Figure 2, illustrates the connect sequence as described below:

- After powering on both HSIC master and slave, master driver is in IDLE bus state.
- Slave monitors the HSIC interface for an IDLE bus state from master.
- Master monitors the HSIC interface for a CONNECT bus state from the slave device.
- Master detects a CONNECT bus state and starts enumeration.

3.2. Configuring HSIC Master/ Slave Mode

The HSIC can be configured as master/slave mode by the end-user application. HSIC interface can be accessible from Linux driver device node (/dev/m2m_drv_cfg) for master/ slave mode configuration.

Example:

```
#define M2M_DRV_CFG_DEV_NAME "/dev/m2m_drv_cfg"
#define M2M_DRV_CFG_MAGIC    't'
#define M2M_DRV_IOCTL_HSIC_GET_MODE    _IOR(M2M_DRV_CFG_MAGIC,0,unsigned int)
#define M2M_DRV_IOCTL_HSIC_SET_MODE    _IOW(M2M_DRV_CFG_MAGIC,1,unsigned int)
/*open device driver node*/
fd = open(M2M_DRV_CFG_DEV_NAME, O_RDWR);

/* Get the status of HSIC mode
   0 - disable HSIC configuration
   1 - Enable HSIC master mode
   2 - Enable HSIC slave mode
*/
ret = ioctl(fd, M2M_DRV_IOCTL_HSIC_GET_MODE, &hsic_mode);

/* Set HSIC to master mode */
hsic_mode = 1;
ret = ioctl(fd, M2M_DRV_IOCTL_HSIC_GET_MODE, &hsic_mode);
if(ret < 0) {
 printf("HSIC mode setting is failed\n");
}
else
{
    /*Manual reboot is required after change HSIC mode*/
   system("reboot");
}
```

4. ETHERNET INTERFACE

The LE910Cx has an embedded Ethernet MAC and only supports SGMII interface.

The embedded Ethernet MAC of LE910Cx supports the following features:

- IEEE 802.3 Ethernet 10/100/1000Mbps, SGMII IF
- SGMII interface can be used using external PHY (SGMII to external PHY)
    - Giga Ethernet PHY can be used by a transceiver chip. For example, Marvell 88EA1512 PHY chip.

4.1. Using SGMII Interface

Before activating the ethernet interface, connect the SGMII interface between LE910Cx and external PHY chip.

The Ethernet interface on the LE910Cx is activated by a shell script (/etc/init.d/start_emac_le) , which can be run by an end-user application.

Example:

```
/etc/init.d/start_emac_le start
```

```
~ #
~ # /etc/init.d/start_emac_le start
[   85.586180] emac start
[   86.057358] libphy: emac-mdio: probed
[   86.069976] qcom-emac 7c40000.qcom,emac (unnamed net_device) (uninitialized): attached PHY driver [Marvell 88E1510]
[   86.084442] android_probe: pm_qos latency not specified 0
[   86.139378] qcom-emac 7c40000.qcom,emac (unnamed net_device) (uninitialized): (mii_bus:phy_addr=7c40000.qcom,ema:00, irq=-1)
[   86.223359] android_probe: pm_qos latency not specified 0
[   86.263246] arp_ignore is set
[   86.352890] qcom-emac 7c40000.qcom,emac eth0: TX queues 1, TX descriptors 512
[   86.371005] qcom-emac 7c40000.qcom,emac eth0: RX queues 1, Rx descriptors 256
[   86.486083] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
done
~ # [   86.523858] USB QCMAP NL IOCTL Snd GETNEIGH Succ
[   86.622528] USB QCMAP NL IOCTL Snd GETNEIGH Succ
[   90.376171] irq 54, desc: cf3df780, depth: 0, count: 0, unhandled: 0
[   90.381520] ->handle_irq():  c026c2b4, msm_gpio_irq_handler+0x0/0x118
[   90.387941] ->irq_data.chip(): c0b30998, 0xc0b30998
[   90.392801] ->action():   (null)
[   90.396007]    IRQ_NOPROBE set
[   90.399046]  IRQ_NOREQUEST set
[   90.402085]   IRQ_NOTHREAD set
[   90.405659] qcom-emac 7c40000.qcom,emac eth0: Link is Up - 1Gbps/Full - flow control rx/tx
[   90.413412] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[   90.451474] QTI:Processing LINK_UP
[   90.455925] QTI:ETH mode
[   90.473396] QTI:LINK_UP message posted
[   90.484509] QTI:Enable mobileap
[   90.498343] QTI:Setup TETHERED link
[   90.674421] device eth0 entered promiscuous mode
[   90.679610] bridge0: port 1(eth0) entered forwarding state
[   90.684224] bridge0: port 1(eth0) entered forwarding state
[   90.721197] QTI:LINK_UP Processed
[   91.120479] ETHERNET Client Mac Address is 28:d2:44:bf:bd:8d
[   94.408277] QCMAP:Ethernet Client IP Addr 192.168.225.33
```

4.1.1. Checking Ethernet Cable Connection Status

Ethernet cable connection status can be checked only when the ethernet PHY and MAC drivers are enabled.

Example:

```
cat /sys/class/net/eth0/carrier
0 : ethernet cable disconnected state
1 : ethernet cable connected state
```

4.1.2. Controlling Ethernet Interface in User Application

The ethernet device driver is provided to control ethernet functions by end-user application. This driver is accessible from Linux driver device node (/dev/m2m_eth).

This driver supports following functions:

- Ethernet mode (LAN mode or WAN mode)
- Ethernet auto connection mode: If auto connection mode is enabled based on the Ethernet mode setting (LAN or WAN mode), a backhaul connection is established or a DHCP client is executed.
- Ethernet disable mode: If disabled mode is set, ethernet driver is disabled.

Note: Ethernet PHY chip should be connected.

Example:

```
#define TELIT_ETH_DEV_NAME  "/dev/m2m_eth"
#define TELIT_ETH_CFG_MAGIC 't'

typedef struct {
/* conn_mode variable *./
int conn_mode;
/* 0: etheret interface is disabled
   2: automatically ethernet interface is enabled and backhaual connection is
established or DHCP client is executed base on ethernet mode setting (LAN
mode or WAN mode).
*/
/* cid variable for PDP Context Identifier*./
  int cid;  // range 1-16
}m2m_conn_mode_type;

#define IOCTL_M2M_ETH_SET_CONN_MODE     _IOW( TELIT_ETH_CFG_MAGIC, 0,
m2m_conn_mode_type )
#define IOCTL_M2M_ETH_GET_CONN_MODE     _IOR( TELIT_ETH_CFG_MAGIC, 1,
m2m_conn_mode_type )
#define IOCTL_M2M_ETH_SET_MODE          _IOW( TELIT_ETH_CFG_MAGIC, 2, int)
#define IOCTL_M2M_ETH_GET_MODE          _IOR( TELIT_ETH_CFG_MAGIC, 3, int)
typedef enum
```

```
{
  ETH_CON_MODE_OFF     = 0,
  ETH_CON_MODE_AUTO    = 2
}eth_con_mode_enum;

typedef enum
{
  ETH_LAN_MODE     = 0,
  ETH_WAN_MODE    = 1
}eth_mode_enum;

int main(int argc, char *argv[])
{
    int fd;
    int result;
    int mode;
    m2m_conn_mode_type m2m_conn_mode = {0,};

    fd = open(TELIT_ETH_DEV_NAME,O_RDWR);
    if(fd  <  0)
  {
       printf("driver open failed \n");
       return -1;
    }
     /* Get current connection mode
    result = ioctl(fd, IOCTL_M2M_ETH_GET_CONN_MODE, &m2m_conn_mode);
    if(result < 0)
    {
          printf("get ethernet connetion mode is failed\n");
    }

    /* Get current ethernet mode
      0 : LAN mode (Deafult)
      1 : WAN mode
    */
    result = ioctl(fd, IOCTL_M2M_ETH_GET_MODE, &mode);
    if(result  <  0)
    {
       printf("ethernet mode setting is failed\n");
    }
```

```
/* change ethernet mode to WAN mode */
m2m_conn_mode.conn_mode = ETH_CON_MODE_OFF;
result = ioctl(fd, IOCTL_M2M_ETH_SET_CONN_MODE, &m2m_conn_mode);
if(result  <  0)
{
  printf("ethernet connetion mode setting is failed\n");
}
mode = ETH_WAN_MODE; // WAN mode
result = ioctl(fd, IOCTL_M2M_ETH_SET_MODE, &mode);
if(result < 0)
{
  printf("ethernet mode setting is failed\n");
}
/* enable auto connection */
m2m_conn_mode.conn_mode = ETH_CON_MODE_AUTO;
result = ioctl(fd, IOCTL_M2M_ETH_SET_CONN_MODE, &m2m_conn_mode);
if(result  <  0)
{
  printf("ethernet connetion mode setting is failed\n");
}

/* change ethernet mode to LAN mode */
 m2m_conn_mode.conn_mode = ETH_CON_MODE_OFF;
result = ioctl(fd, IOCTL_M2M_ETH_SET_CONN_MODE, &m2m_conn_mode);
if(result  <  0)
{
  printf("ethernet connetion mode setting is failed\n");
}
mode = ETH_LAN_MODE; // LAN mode
result = ioctl(fd, IOCTL_M2M_ETH_SET_MODE, &mode);
if(result < 0)
{
  printf("ethernet mode setting is failed\n");
}
m2m_conn_mode.conn_mode = ETH_CON_MODE_AUTO;
m2m_conn_mode.cid = 1;
result = ioctl(fd, IOCTL_M2M_ETH_SET_CONN_MODE, &m2m_conn_mode);
if(result  <  0)
{
  printf("ethernet connetion mode setting is failed\n");
```

```
    }

  close(fd)
  return 0;
}
```

4.1.3. Enabling/Disabling the "CLK125" of External Marvell PHY (88E1512/5)

If you use an external PHY instead of a Marvell PHY, you can control the "CLK125 (Page 2, Reg 16 bit 2)".

Value	Description
0 (Default)	Enable internally generated 125MHz clock
1	Disable internally generated 125MHz clock

Table 3: Enable/Disable CLK125

For example,

- To enable the CLK125,

```
# echo 0 > /data/marvell_clk
```

  If a value of the "/data/marvell_clk" is set to 0 as above, 125MHz clock is enabled.

- To disable the CLK125,

```
# echo 1 > /data/marvell_clk
```

  If a value of the "/data/marvell_clk" is set to 1 as above, 125MHz clock is disabled.

Note: To control CLK125, it must be set before the ethernet interface is activated.

The setting is not retained after a firmware update, but it is retained after a FOTA update.

This feature is only available for the LE910C1-EU (4G+2G) variant.

5. GPIO Interface

LE910Cx provides 10 GPIOs and 8 UART pins, which can be configured as Input and Output through a Linux device driver.

These GPIO pins allow your application to control external hardware directly from the GPIO pins, requiring little or no additional hardware.

The LE910Cx supports the following GPIO pins:

Pin Number	GPIO/UART Pins
1	GPIO1
2	GPIO2
3	GPIO3
4	GPIO4
5	GPIO5
6	GPIO6
7	GPIO7
8	GPIO8
9	GPIO9
10	GPIO10
20	DCD
21	CTS
22	RI
23	DSR
24	DTR
25	RTS
26	RXD
27	TXD

Table 4: LE910Cx Supported GPIO Pins

To use UART pins as GPIO, use the #V24CFG command to set them to GPIO mode. For details refer to section 8.1 Using #V24CFG Command.

5.1. Using GPIO Interface

The GPIO device driver is provided to allow the common use of GPIOs in various LE910Cx hardware configurations.

The GPIOs can be used externally by the end-user application. The GPIO interface is accessible from Linux driver device node (/dev/m2m_gpio).

The following is the list of the supported GPIO I/F parameters:

Parameter	Description
<mode>	Set GPIO modes: 0 - Clear the use of GPIO 1 - Set GPIO direction 2 - Set GPIO output value 3 - Read GPIO value
<gpio>	GPIO pin number: (TGPIO)1-10, (UART)20-27
<dir>	GPIO pin direction: 0 - Pin direction is INPUT 1 - Pin direction is OUTPUT
<val>	Its meaning depends on <dir> setting: 0 - Output pin set to 0 (Low) if <dir>=1 - OUTPUT - Input pin set to 0 (Pull-down) if <dir>=0 - INPUT (default) 1 - Output pin set to 1 (High) if <dir>=1 - OUTPUT - Input pin set to 1 (Pull-up) if <dir>=0 - INPUT 2 - Input pin set to 2 (No-Pull) if <dir>=0 - INPUT

Table 5: Supported GPIO I/F Parameters

Example:

```
#define GPIO_DEV_PATH        "/dev/m2m_gpio"

/* Parameters to be passed through  IOCTL */
 typedef struct {
   unsigned int m2m_gpio_num;
   unsigned int m2m_gpio_dir;
   unsigned int m2m_gpio_val;
 }m2m_gpio_info;

#define M2M_GPIO_MAGIC       'g'
#define IOCTL_M2M_APP_GPIO_CLR         _IOW( M2M_APP_GPIO_MAGIC, 0,
m2m_gpio_info )
#define IOCTL_M2M_APP_GPIO_SET_DIR     _IOW( M2M_APP_GPIO_MAGIC, 1,
m2m_gpio_info )
#define IOCTL_M2M_APP_GPIO_SET_VAL     _IOW( M2M_APP_GPIO_MAGIC, 2,
m2m_gpio_info )
#define IOCTL_M2M_APP_GPIO_GET_VAL     _IOW( M2M_APP_GPIO_MAGIC, 3,
m2m_gpio_info )

#define MAX_DEFIEND_TGPIO_NUM 10
#define MIN_UART_GPIO 20
#define MAX_UART_GPIO 27
```

```
/* GPIO value parameters for output */
enum
{
  M2M_APP_GPIO_OUT_LOW = 0,
  M2M_APP_GPIO_OUT_HIGH,
  M2M_APP_GPIO_OUT_MAX
};
/* GPIO pull parameters for input */
enum
{
  M2M_APP_GPIO_IN_PD = 0,
  M2M_APP_GPIO_IN_PU,
  M2M_APP_GPIO_IN_NP,
  M2M_APP_GPIO_IN_MAX
};
/* GPIO direction parameters */
enum
{
  M2M_APP_GPIO_DIR_IN = 0,
  M2M_APP_GPIO_DIR_OUT,
  M2M_APP_GPIO_DIR_MAX
};
/* GPIO command parameters */
enum
{
  M2M_APP_GPIO_MODE_CLR = 0,
  M2M_APP_GPIO_MODE_SET_DIR,
  M2M_APP_GPIO_MODE_SET_VAL,
  M2M_APP_GPIO_MODE_GET_VAL,
  M2M_APP_GPIO_MODE_MAX
};
/* GPIO command parameters */
enum
{
  M2M_APP_GPIO_MODE_CLR = 0,
  M2M_APP_GPIO_MODE_SET_DIR,
  M2M_APP_GPIO_MODE_SET_VAL,
  M2M_APP_GPIO_MODE_GET_VAL,
  M2M_APP_GPIO_MODE_MAX
};
```

```
 int main(int argc, char *argv[])
{
  int dev = 0;
  int ret = -1;
  m2m_gpio_info *m2m_gpio;

  if(atoi(argv[1]) >= M2M_APP_GPIO_MODE_MAX)
  {
    perror("[GPIO] Mode Parameter out of range \n");
    return -1;
  }

  dev = open(GPIO_DEV_PATH, O_RDWR);
  if(dev < 0)
  {
    perror("[GPIO] driver open failed \n");
    return -1;
  }

  m2m_gpio = (m2m_gpio_info *)malloc(sizeof(m2m_gpio_info));
  memset(m2m_gpio, 0x00, sizeof(m2m_gpio_info));

  switch(atoi(argv[1]))
  {
    /* When the use of GPIO is completed, it should be cleared and made
available to other devices */
    case M2M_GPIO_MODE_CLR:
      if(((atoi(argv[2]) > MAX_DEFINED_TGPIO_NUM) && (atoi(argv[2]) <
MIN_UART_GPIO))
        || (atoi(argv[2]) > MAX_UART_GPIO))
      {
        perror("[GPIO] GPIO parameter out of range \n");
        return -1;
      }
      m2m_gpio->m2m_gpio_num = atoi(argv[2]);

      ret = ioctl(dev, IOCTL_M2M_APP_GPIO_CLR, m2m_gpio);
      if(ret)
      {
        perror("[GPIO] ioctl control failure \n");
```

```
            return ret;
          }
          break;

        /* Direction should be set to Input (with pull) or Output to control the
GPIO */
        case M2M_GPIO_MODE_SET_DIR:
          if((((atoi(argv[2]) > MAX_DEFINED_TGPIO_NUM) && (atoi(argv[2]) <
MIN_UART_GPIO))
            || (atoi(argv[2]) > MAX_UART_GPIO))
            ||(atoi(argv[3]) >= M2M_APP_GPIO_DIR_MAX))
          {
            perror("[GPIO] GPIO parameter out of range \n");
            return -1;
          }

          m2m_gpio->m2m_gpio_num = atoi(argv[2]);
          m2m_gpio->m2m_gpio_dir = atoi(argv[3]);
          if(argv[4])
          {
            if(((atoi(argv[3]) == M2M_APP_GPIO_DIR_IN) && ((atoi(argv[4])) >=
M2M_APP_GPIO_IN_MAX))
            || ((atoi(argv[3]) == M2M_APP_GPIO_DIR_OUT) && ((atoi(argv[4])) >=
M2M_APP_GPIO_OUT_MAX)))
            {
                perror("[GPIO] GPIO parameter out of range \n");
                return -1;
            }
            else
            {
                m2m_gpio->m2m_gpio_val = atoi(argv[4]);
            }
          }
          else
          {
            if(atoi(argv[3]) == M2M_APP_GPIO_DIR_IN)
            {
                m2m_gpio->m2m_gpio_val = M2M_APP_GPIO_IN_PD;   // default pull-
down
            }
            else
            {
```

```
            perror("[GPIO] Invalid parameter \n");
            return -1;
          }
        }

        ret = ioctl(dev, IOCTL_M2M_APP_GPIO_SET_DIR, m2m_gpio);
        if(ret)
        {
            perror("[GPIO] ioctl control failure \n");
            return ret;
        }
        break;

      /* When setting GPIO's output  values (High / Low), direction should be
set to OUTPUT first. */
      case M2M_GPIO_MODE_SET_VAL:
        if((((atoi(argv[2]) > MAX_DEFINED_TGPIO_NUM) && (atoi(argv[2]) <
MIN_UART_GPIO))
          || (atoi(argv[2]) > MAX_UART_GPIO))
          || (atoi(argv[3]) >= M2M_APP_GPIO_OUT_MAX))
        {
          perror("[GPIO] GPIO parameter out of range \n");
          return -1;
        }

        m2m_gpio->m2m_gpio_num = atoi(argv[2]);
        m2m_gpio->m2m_gpio_dir = M2M_APP_GPIO_DIR_OUT;
        m2m_gpio->m2m_gpio_val = atoi(argv[3]);

        ret = ioctl(dev, IOCTL_M2M_APP_GPIO_SET_VAL, m2m_gpio);
        if(ret)
        {
          perror("[GPIO] ioctl control failure \n");
          return ret;
        }
        break;

      /* Read the current GPIO pin status */
      case M2M_GPIO_MODE_GET_VAL:
        if(((atoi(argv[2]) > MAX_DEFINED_TGPIO_NUM) && (atoi(argv[2]) <
MIN_UART_GPIO))
```

```
          || (atoi(argv[2]) > MAX_UART_GPIO))
        {
          perror("[GPIO] GPIO parameter out of range \n");
          return -1;
        }

        m2m_gpio->m2m_gpio_num = atoi(argv[2]);
        m2m_gpio->m2m_gpio_dir = M2M_GPIO_DIR_IN;

        ret = ioctl(dev, IOCTL_M2M_APP_GPIO_GET_VAL, m2m_gpio);
        if(ret)
        {
          perror("[GPIO] ioctl control failure \n");
          return ret;
        }
        break;

      default:
        break;
    }

    free(m2m_gpio);
    close(dev);
    return ret;
}
```

5.2. Using GPIO Interrupt

The GPIO-keys module allows a Linux-based application, to listen to GPIO interrupts. This can be accomplished using a GPIO 1-10.

Application can then listen to **"/dev/input/event1"** to get the interrupt and the interrupt data.

Several GPIOs are able to wake up the system from sleep. When using such a GPIO with the GPIO-KEYS driver, any interrupt on this line will wake the system. Using a GPIO that is not capable of waking up the system with the GPIO-KEYS driver will PREVENT THE SYSTEM FROM GOING INTO SLEEP (the logic is very simple: if there is an interrupt pending on a non-wakeup capable GPIO, do not go to sleep).

The GPIO-Keys module has three parameters:

- **tgpios** – An array of tgpios to listen on. For example, tgpios=4,5 causes the driver to listen to tgpio4 and tgpio5.
- **pull_arr** – An optional array of pull settings to apply to each tgpio used. The following options are available:
  - 0 – No Pull
  - 1 – Pull Up
  - 2 – Pull Down
  - 3 – Default
- **debounce_interval** – An optional array of debounce intervals to apply to each tgpio used. The value should be greater than or equal to 0, for example debounce_interval = 10 means 1ms. The default value is 15.

Insert command for the GPIO-Keys module:

```
"insmod /data/gpio-keys tgpios=<GPIO>[,<GPIO>,,,] pull_arr=<pull>[,<pull>,,,]
[debounce_interval=<ms>[,<ms>,,,]]"
```

Remove command for the GPIO-Keys module:

```
"rmmod gpio-keys"
```

Example:

To start the gpio-keys driver listen on tgpio4 (no pull) and tgpio5 (pull up), use the following command:

```
"insmod /data/gpio-keys tgpios=4,5 pull_arr=0,1"
```

And if the gpio-keys driver listen on tgpio4 (pull up), use the following command:

```
"insmod /data/gpio-keys tgpios=4 pull_arr=1"
```

To set debounce interval of 0.7ms on tgpio4 and 1ms on tgpio5 use the following command:

```
"insmod /data/gpio-keys tgpios=4,5 pull_arr=0,1 debounce_interval=7,10"
```

Note: The number of tgpios parameters must match the number of pull_arr parameters, otherwise pull_arr is totally ignored.

With debounce_interval set to 0, usually the average of detectable interrupts in 1s is around 1600.

Note: The following GPIOs are wake up capable (All other GPIOs are not wakeup capable):

- GPIO1
- GPIO5
- GPIO8

Warning: Some GPIOs (GPIO1, GPIO5 ~ 9) should not be pulled high externally (by the carrier board) during module power on procedure. Pulling those pads high during module power up might lead to unwanted/non-operational boot mode.

Refer Hardware User Guide for more details.

Note: GPIO1 and GPIO8 each have "SLED" and "SWREADYEN" functions by default, so in order to use the GPIO interface, the functions should be disabled through AT command first.

m2m_gpio and GPIO-Keys cannot use the same GPIO at the same time, but in the case of GPIO with interrupt set by GPIO-Keys, it is possible to read the value of GPIO through m2m_gpio.

6. SPI INTERFACE

LE910Cx provides a 4-wire SPI (Serial Peripheral Interface) and the H/W Pins of SPI are shared with Aux UART, so SPI and Aux UART cannot be used simultaneously.

LE910Cx provides the device driver node(/dev/m2m_drv_cfg) to switch from Aux UART to SPI or from SPI to Aux UART and this device driver node is used to configure SPI CS, interrupt, and slave ready GPIO by end-user application.

SPI interrupt and SPI slave ready GPIO are optional function.

The table below lists the supported GPIO pins for SPI CS, SPI interrupt or SPI slave ready on LE910Cx.

GPIO Pins	Descriptions
1	GPIO1
2	GPIO2
3	GPIO3
4	GPIO4
5	GPIO5
6	GPIO6
7	GPIO7
8	GPIO8
9	GPIO9
10	GPIO10

Table 6: LE910Cx Supported GPIO Pins for SPI Interface

6.1. Switching from SPI to Aux UART or from Aux UART to SPI

The driver device node (/dev/m2m_drv_cfg) can be used to switch from SPI to Aux UART or from Aux UART to SPI by the end-user application.

Example:

```
#define M2M_DRV_CFG_DEV_NAME  "/dev/m2m_drv_cfg"
#define M2M_DRV_CFG_MAGIC 't'
#define M2M_DRV_IOCTL_GET_SPI_STATUS  _IOR(M2M_DRV_CFG_MAGIC,  2, unsigned int)
#define M2M_DRV_IOCTL_SET_SPI_STATUS  _IOW(M2M_DRV_CFG_MAGIC,  3, unsigned int)


int main(int argc, char *argv[])
```

```
{
int fd = 0;
unsigned int spi_status = 0;

  fd = open(M2M_DRV_CFG_DEV_NAME, O_RDWR);
  if(fd < 0)
  {
    printf("%s driver open failed \n", M2M_DRV_CFG_DEV_NAME);
    return -1;
  }
/*Get SPI status 1: Enable SPI, 0: Disable SPI*/
if(ioctl(fd, M2M_DRV_IOCTL_GET_SPI_STATUS, &spi_status) < 0)
{
     printf("Unable to get current status\n");
}
spi_status = 1; //1: Switch from Aux UART to SPI | 0:  Switch from SPI to Aux
UART.
if(ioctl(fd, M2M_DRV_IOCTL_SET_SPI_STATUS, &spi_status) < 0)
{
   printf("Unable to set status\n");
}
else{
   system("reboot");
}

close(fd);

return 0;
```

6.2. Configuring SPI to Support Multiple CS for Multiple Slave Devices

The driver device node (/dev/m2m_drv_cfg) can be used to support multiple slave devices by the end-user application.

When the multiple SPI CS pins are configured by the end-user application, the end-user application must execute “reboot”. From the next boot-up, LE910Cx configures the multiple CS pins and creates SPI device driver nodes (/dev/spievB.0, /dev/spidevB.1, and /dev/spidevB.2).

If SPI CS pins are not configured, the SPI master of LE910Cx controls the dedicated SPI_CS_pin for SPI device driver nodes (/dev/spievB.0, /dev/spidevB.1, and /dev/spidevB.2). If you only want to use one SPI slave device, use "/dev/spidevB.0."

The end user should check SPI device driver nodes.

Example:

```
#define M2M_DRV_CFG_DEV_NAME  "/dev/m2m_drv_cfg"
#define M2M_DRV_CFG_MAGIC 't'
#define M2M_DRV_IOCTL_GET_SPI_STATUS  _IOR(M2M_DRV_CFG_MAGIC,  2, unsigned int)
#define M2M_DRV_IOCTL_SET_SPI_STATUS  _IOW(M2M_DRV_CFG_MAGIC,  3, unsigned int)
#define M2M_DRV_IOCTL_GET_SPI_CFG_INFO     _IOR( M2M_DRV_CFG_MAGIC, 4, m2m_spi_info_type[3] )
#define M2M_DRV_IOCTL_SET_SPI_CFG_INFO     _IOW( M2M_DRV_CFG_MAGIC, 5, m2m_spi_info_type[3] )

/* Parameters to be passed through  IOCTL */
typedef struct {
   unsigned int cs_gpio;
   unsigned int int_gpio;
   unsigned int slave_ready_gpio;
}m2m_spi_info_type;



int main(int argc, char *argv[])
{
int fd = 0;
m2m_spi_info_type m2m_spi_info[3]={0,};
unsigned int spi_status = 0;

  fd = open(M2M_DRV_CFG_DEV_NAME, O_RDWR);
  if(fd < 0)
  {
    printf("%s driver open failed \n", M2M_DRV_CFG_DEV_NAME);
    return -1;
  }
if(ioctl(fd, M2M_DRV_IOCTL_GET_SPI_STATUS, &spi_status) < 0)
{
    printf("Unable to get spi_status\n");
  }
 if(spi_status == 0)
```

```
    spi_status = 1; // If SPI is enabled, from next boot-up, SPI device driver
nodes are created.

 if(ioctl(fd, M2M_DRV_IOCTL_SET_SPI_STATUS, &spi_status) < 0)
 {
   printf("unable to set spi_status\n");
 }

/*Get current SPI configuration information*/
 if(ioctl(fd, M2M_DRV_IOCTL_GET_SPI_CFG_INFO, &m2m_spi_info) < 0)
 {
   printf("unable to get SPI configuration information\n");
 }

/*Set SPI configuration information*/
 m2m_spi_info[0].cs_gpio = 0;
 m2m_spi_info[0].int_gpio = 0;
 m2m_spi_info[0].slave_ready_gpio = 0;
 m2m_spi_info[1].cs_gpio = 8;
 m2m_spi_info[1].int_gpio = 0;
 m2m_spi_info[1].slave_ready_gpio = 0;
 m2m_spi_info[2].cs_gpio = 9;
 m2m_spi_info[2].int_gpio = 0;
 m2m_spi_info[2].slave_ready_gpio = 0;
 if(ioctl(fd, M2M_DRV_IOCTL_SET_SPI_CFG_INFO, &m2m_spi_info) < 0)
 {
   printf("unable to set SPI configuration information\n");
 }
else
{
   system("reboot");
}
close(fd);
return 0;
```

6.3. Configuring SPI to Support Multiple Slave Devices with Interrupt

The device driver node(/dev/m2m_drv_cfg) can be used to support multiple slave devices with interrupt by the end-user application.

If SPI interrupts are configured by the end-user application, the end-user application must execute “reboot”. From the next boot-up, LE910Cx configures SPI interrupts with “IRQF_TRIGGER_RISING | IRQF_TRIGGER_FALLING” properties.

Example:

```
#define M2M_DRV_CFG_DEV_NAME  "/dev/m2m_drv_cfg"
#define M2M_DRV_CFG_MAGIC 't'
#define M2M_DRV_IOCTL_GET_SPI_STATUS  _IOR(M2M_DRV_CFG_MAGIC,  2, unsigned
int)
#define M2M_DRV_IOCTL_SET_SPI_STATUS  _IOW(M2M_DRV_CFG_MAGIC,  3, unsigned
int)
#define M2M_DRV_IOCTL_GET_SPI_CFG_INFO     _IOR( M2M_DRV_CFG_MAGIC, 4,
m2m_spi_info_type[3] )
#define M2M_DRV_IOCTL_SET_SPI_CFG_INFO     _IOW( M2M_DRV_CFG_MAGIC, 5,
m2m_spi_info_type[3] )

/* Parameters to be passed through  IOCTL */
typedef struct {
   unsigned int cs_gpio;
   unsigned int int_gpio;
   unsigned int slave_ready_gpio;
}m2m_spi_info_type;


int main(int argc, char *argv[])
{
int fd = 0;
m2m_spi_info_type m2m_spi_info[3]={0,};
unsigned int spi_status = 0;

  fd = open(M2M_DRV_CFG_DEV_NAME, O_RDWR);
  if(fd < 0)
  {
    printf("%s driver open failed \n", M2M_DRV_CFG_DEV_NAME);
    return -1;
  }
if(ioctl(fd, M2M_DRV_IOCTL_GET_SPI_STATUS, &spi_status) < 0)
{
    printf("Unable to get spi_status\n");
  }
 if(spi_status == 0)
```

```
    spi_status = 1; // If SPI is enabled, from next boot-up, SPI device driver
nodes are created.

 if(ioctl(fd, M2M_DRV_IOCTL_SET_SPI_STATUS, &spi_status) < 0)
 {
   printf("unable to set spi_status\n");
 }

/*Get current SPI configuration information*/
 if(ioctl(fd, M2M_DRV_IOCTL_GET_SPI_CFG_INFO, &m2m_spi_info) < 0)
 {
   printf("unable to get SPI configuration information\n");
 }

/*Set SPI configuration information*/
 m2m_spi_info[0].cs_gpio = 0;
 m2m_spi_info[0].int_gpio = 2;
 m2m_spi_info[0].slave_ready_gpio = 0;
 m2m_spi_info[1].cs_gpio = 8;
 m2m_spi_info[1].int_gpio = 3;
 m2m_spi_info[1].slave_ready_gpio = 0;
 m2m_spi_info[2].cs_gpio = 9;
 m2m_spi_info[2].int_gpio = 4;
 m2m_spi_info[2].slave_ready_gpio = 0;
 if(ioctl(fd, M2M_DRV_IOCTL_SET_SPI_CFG_INFO, &m2m_spi_info) < 0)
 {
   printf("unable to set SPI configuration information\n");
 }
else
{
   system("reboot");
}

close(fd);

return 0;
```

6.3.1. Getting SPI Interrupt in Application Layer

Example:

```
#include <stdio.h>
```

```
#include <stdint.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/types.h>
#include <sys/types.h>
#include <ctype.h>
#include <getopt.h>
#include <time.h>
#include <linux/spi/spidev.h>
#include <poll.h>
#include <linux/types.h>
#include <linux/ioctl.h>

int main(int argc, char *argv[])
{
   int fd = 0;
   int ret;
   uint8_t mode = 0; // please set mode according to slave device
environment.
   uint32_t speed = 50000000; // please set speed according to slave device
environment.
   uint8_t bits_per_word = 8;
   struct pollfd poll_fds[1];
   if (access("/sys/devices/78b9000.spi/spi_master", F_OK) != 0)
   {
      fd = open("/dev/spidev1.0", O_RDWR);
      if(fd < 0)
      {
        printf("spidev1.0 driver open failed \n");
        return -1;
      }
   }
   else{
      fd = open("/dev/spidev2.0", O_RDWR);
      if(fd < 0)
      {
        printf("spidev2.0 driver open failed \n");
```

```
            return -1;
        }
    }

    /*
    * spi mode
    */
    ret = ioctl(fd, SPI_IOC_WR_MODE, &mode);
    if (ret == -1) printf("cant set WR spi mode");

    ret = ioctl(fd, SPI_IOC_RD_MODE, &mode);
    if (ret == -1) printf("can't set RD spi mode");

     /*
         * bits per word
     */
    ret = ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &bits_per_word);
    if (ret == -1) printf("can't set WR bits per word");
    ret = ioctl(this->fd, SPI_IOC_RD_BITS_PER_WORD, &bits_per_word);
    if (ret == -1) printf("can't set RD bits per word");

     /*
     * max speed hz
    */
    ret = ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed);
    if (ret == -1) printf"can't WR set max speed hz");

    ret = ioctl(fd, SPI_IOC_RD_MAX_SPEED_HZ, &speed);
    if (ret == -1) printf("can't RD set max speed hz");
    /*
     * Waiting SPI interrupt singal using poll function.
    */
    poll_fds[0].fd = fd;
    poll_fds[0].events = POLLIN | POLLRDNORM;
    while (1)
    {
        ret = poll(poll_fds, 1, -1);
        if(ret > 0){
              printf(" Interrupt is happened\n");
               // Read SPI data. If SPI read is called, SPI driver clears the
poll event.
```

```
                break;
            }
    }
    close(fd);
return 0;
}
```

6.4. Configuring SPI to Support Multiple Slave Ready Signal

The device driver node(/dev/m2m_drv_cfg) can be used to support multiple slave devices with interrupt by the end-user application.

Whenever an SPI slave device is not ready to transmit data on SPI bus, it turns GPIO output to high state. When LE910Cx receives a high state from an SPI slave device, it waits for 5 sec for low state from SPI slave device. If SPI slave device does not turn to a GPIO low state, error occur in LE910Cx during SPI read/ write operation.

Example:

```
#define M2M_DRV_CFG_DEV_NAME  "/dev/m2m_drv_cfg"
#define M2M_DRV_CFG_MAGIC 't'
#define M2M_DRV_IOCTL_GET_SPI_STATUS  _IOR(M2M_DRV_CFG_MAGIC,  2, unsigned
int)
#define M2M_DRV_IOCTL_SET_SPI_STATUS  _IOW(M2M_DRV_CFG_MAGIC,  3, unsigned
int)
#define M2M_DRV_IOCTL_GET_SPI_CFG_INFO     _IOR( M2M_DRV_CFG_MAGIC, 4,
m2m_spi_info_type[3] )
#define M2M_DRV_IOCTL_SET_SPI_CFG_INFO     _IOW( M2M_DRV_CFG_MAGIC, 5,
m2m_spi_info_type[3] )

/* Parameters to be passed through  IOCTL */
typedef struct {
   unsigned int cs_gpio;
   unsigned int int_gpio;
   unsigned int slave_ready_gpio;
}m2m_spi_info_type;


int main(int argc, char *argv[])
{
int fd = 0;
m2m_spi_info_type m2m_spi_info[3]={0,};
unsigned int spi_status = 0;
```

```
    fd = open(M2M_DRV_CFG_DEV_NAME, O_RDWR);
    if(fd < 0)
    {
      printf("%s driver open failed \n", M2M_DRV_CFG_DEV_NAME);
      return -1;
    }
if(ioctl(fd, M2M_DRV_IOCTL_GET_SPI_STATUS, &spi_status) < 0)
{
      printf("Unable to get spi_status\n");
    }
 if(spi_status == 0)
   spi_status = 1; // If SPI is enabled, from next boot-up, SPI device driver
nodes are created.

 if(ioctl(fd, M2M_DRV_IOCTL_SET_SPI_STATUS, &spi_status) < 0)
 {
   printf("unable to set spi_status\n");
 }

/*Get current SPI configuration information*/
 if(ioctl(fd, M2M_DRV_IOCTL_GET_SPI_CFG_INFO, &m2m_spi_info) < 0)
 {
   printf("unable to get SPI configuration information\n");
 }

/*Set SPI configuration information*/
   m2m_spi_info[0].cs_gpio = 0;
   m2m_spi_info[0].int_gpio = 2;
   m2m_spi_info[0].slave_ready_gpio = 5;
   m2m_spi_info[1].cs_gpio = 8;
   m2m_spi_info[1].int_gpio = 3;
   m2m_spi_info[1].slave_ready_gpio = 6;
   m2m_spi_info[2].cs_gpio = 9;
   m2m_spi_info[2].int_gpio = 4;
   m2m_spi_info[2].slave_ready_gpio = 7;

 if(ioctl(fd, M2M_DRV_IOCTL_SET_SPI_CFG_INFO, &m2m_spi_info) < 0)
 {
   printf("unable to set SPI configuration information\n");
 }
else
```

```
{
   system("reboot");
}

close(fd);

return 0;
```

7. SD/MMC CARD INTERFACE

LE910Cx provides an SD port that supports the SD3.0 specification and can be used with standard SD/MMC memory cards.

7.1. Detecting/Mounting of SD/MMC Memory Card

1. When an SD/MMC memory card is inserted, the device node is created automatically as shown below.

```
~ # [ 2782.212319] mmc1: new high speed SD card at address 1234
[ 2782.217521] mmcblk0: mmc1:1234 SA01G 942 MiB
[ 2782.226550]  mmcblk0:
[ 2782.244978] android_probe: pm_qos latency not specified 0

~ # ls -al /dev/mmc*
brw-rw----    1 root     disk      179,   0 Jan  6 00:46 /dev/mmcblk0
~ #
```

2. Once the device node appears, run the below command from an end-user application or from the adb shell.

Example:

```
mount -t vfat /dev/mmcblk0 /mnt/sdcard
```

3. Verify that the file system has been mounted (refer to the last line in the below output):

```
/ # mount
rootfs on / type rootfs (rw)
proc on /proc type proc (rw,relatime)
none on /sys type sysfs (rw,relatime)
/dev/ubiblock0_0 on / type squashfs (ro,relatime)
/dev/ubiblock0_2 on /sdata type squashfs (ro,relatime)
tmpfs on /dev type tmpfs (rw,relatime,size=64k,nr_inodes=19717,mode=755)
devpts on /dev/pts type devpts (rw,relatime,gid=5,mode=620)
tmpfs on /run type tmpfs (rw,nosuid,nodev,size=78868k,nr_inodes=19717,mode=755)
tmpfs on /var/volatile type tmpfs (rw,relatime,size=78868k,nr_inodes=19717)
tmpfs on /var/lib type tmpfs (rw,relatime,size=78868k,nr_inodes=19717)
ubi0:usrfs on /data type ubifs (rw,relatime,bulk_read,chk_data_crc)
ubi0:cachefs on /cache type ubifs (rw,relatime,bulk_read,chk_data_crc)
/dev/ubi1_0 on /firmware type ubifs (ro,relatime,bulk_read,chk_data_crc)
none on /sys/kernel/config type configfs (rw,relatime)
adb on /dev/usb-ffs/adb type functionfs (rw,relatime)
/dev/mmcblk0 on /mnt/sdcard type vfat (rw,relatime,fmask=0022,dmask=0022,codepage=437,iocharset=iso8859-1,shortname=mixed,errors=remount-ro)
```

4. To unmount the SD/MMC memory card from /mnt/sdcard run the below command:

Example:

```
umount /mnt/sdcard
```

```
/ # umount /mnt/sdcard
/ #
/ #
```

5. Remove the SD/MMC memory card from the card slot.

8. UART INTERFACE

LE910Cx supports two UART interfaces. Main UART pins include TX data (TXD), RX data (RXD), Request To Send (RTS), Clear To Send (CTS), Data Terminal Ready (DTR), Data Carrier Detect (DCD), and Ring Indicator (RI).

Note: The SPI hardware pins are shared with Aux UART, hence SPI and Aux UART cannot be used simultaneously.

The following functions are supported by the UART interface:

- AT#V24CFG and AT#V24 command
- AT#PORTCFG command

Refer to AT commands Reference Guide for more details.

8.1. Using #V24CFG Command

#V24CFG command is used to configure the serial interface pins as GPIO.

To support V24CFG command, a device driver is provided, which can be used externally by the end-user application. The device driver is accessible from Linux driver device node (/dev/m2m_drv_cfg).

Example:

```
#define M2M_DRV_CFG_DEV_NAME  "/dev/m2m_drv_cfg"
#define M2M_DRV_CFG_MAGIC 't'
#define M2M_DRV_IOCTL_GET_V24_CFG      _IOR( M2M_DRV_CFG_MAGIC, 6, unsigned
int [8] )
#define M2M_DRV_IOCTL_SET_V24_CFG      _IOW( M2M_DRV_CFG_MAGIC, 7, unsigned
int [8] )

int main(int argc, char *argv[])
{
    int fd = 0;
    int result = 0
    unsigned int v24cfg_mode[8]={0,};

      fd = open(M2M_DRV_CFG_DEV_NAME, O_RDWR);
      if(fd < 0)
     {
         printf("%s driver open failed \n", M2M_DRV_CFG_DEV_NAME);
         return -1;
    }
```

```
    //Get the current setting information
    result = ioctl(fd, M2M_DRV_IOCTL_GET_V24_CFG, &v24cfg_mode);
    if(result < 0)
    {
        printf("Failed read V24CFG info\n");
    }
      /*
        v24cfg_mode[0] : DCD
        v24cfg_mode[1] : CTS
        v24cfg_mode[2] : RI
        v24cfg_mode[3] : DSR
        v24cfg_mode[4] : DTR
        v24cfg_mode[5] : RTS
        v24cfg_mode[6] : RXD
        v24cfg_mode[7] : TXD

        if the value for each index in the array is 0,1 or 2:
           0 : AT commands serial port mode
           1 : GPIO mode  Pins directly controlled by #V24 command
           2 : GPIO kernel mode  Pins directly controlled by kernel GPIO
driver.
      */
      v24cfg_mode[0] = 2;
      v24cfg_mode[1] = 2;
      v24cfg_mode[2] = 2;
      v24cfg_mode[3] = 2;
      v24cfg_mode[4] = 2;
      v24cfg_mode[5] = 2;
      v24cfg_mode[6] = 2;
      v24cfg_mode[7] = 2;
      result = ioctl(fd, M2M_DRV_IOCTL_SET_V24_CFG, &v24cfg_mode);
      if(result < 0)
      {
        printf("Failed V24CFG setting\n");
      }
      else{
        printf("V24CFG setting is succeeded\n");
        system("reboot");  //module must be reboot, the pins configuration is
applied next power cycle
      }
      close(fd);
```

```
        return 0;
}
```

8.2. Using #PORTCFG Command

#PORTCFG supports the following variants.

Variants	Descriptions
Variant 0	USIF0, USB0, and USB1 are connected to AT port.
Variant 3	USIF0, USIF1, and USB0 are connected to AT port.
Variant 8	USB0 and USB1 are connected to AT port.
Variant 11	USIF0, USB0 and USB1 are connected to AT port. USIF1 is used for NMEA Sentences.
Variant 14(default)	USIF0, USIF1, USB0 and USB1 are connected to AT port.
Variant 15	USIF0, USB0 and USB1 are connected to AT port. USIF1 is connected to console port.
Variant 16	USIF0, USB0 and USB1 are connected to AT port. USIF1 is used for external BT UART supporting.

Table 6 : #PORTCFG Command

To support #PORTCFG command, device driver is provided, and the device driver can be used externally by the end-user application. The device driver is accessible from Linux driver device node(/dev/m2m_drv_cfg).

Example:

```
#define M2M_DRV_CFG_DEV_NAME  "/dev/m2m_drv_cfg"
#define M2M_DRV_CFG_MAGIC 't'
#define M2M_DRV_IOCTL_GET_PORTCFG  _IOR( M2M_DRV_CFG_MAGIC, 8,
m2m_portcfg_info_type )
#define M2M_DRV_IOCTL_SET_PORTCFG  _IOW( M2M_DRV_CFG_MAGIC, 9, unsigned int)

typedef struct {
  unsigned int act_variant;
  unsigned int req_variant;
}m2m_portcfg_info_type;

int main(int argc, char *argv[])
{
    int fd = 0;
    int result = 0
    m2m_portcfg_info_type m2m_portcfg_info ={0,};
    unsigned int req_variant = 0;
```

```
        fd = open(M2M_DRV_CFG_DEV_NAME, O_RDWR);
        if(fd < 0)
       {
           printf("%s driver open failed \n", M2M_DRV_CFG_DEV_NAME);
           return -1;
      }
      //Get the current setting information
      result = ioctl(fd, M2M_DRV_IOCTL_GET_PORTCFG, &m2m_portcfg_info);
      if(result < 0)
     {
          printf("Failed read PORTCFG info\n");
     }
/*
    if the value of m2m_portcfg_info.act_variant is 0,3,8,11,14,15 or 16.
    0: USIF0, USB0, and USB1 are connected to AT port
    3: USIF0, USIF1, and USB0 are connected to AT port
    8: USB0 and USB1 are connected to AT port.
    11: USIF0, USB0 and USB1 are connected to AT port and USIF1 is used for
NMEA Setences.
    14: USIF0, USIF1, USB0 and USB1 is connected to AT port.
    15: USIF0, USB0 and USB1 are connected to AT port and USIF1 is connected
to console port.
    16: USIF0, USB0 and USB1 are connected to AT port and USIF1 is used for
external BT UART supporting
 */
  //Set the variant of #PORTCFG
   req_variant = 15;
   result = ioctl(fd, M2M_DRV_IOCTL_SET_PORTCFG, &req_variant);
   if(result < 0)
   {
        printf("Failed PORTCFG setting\n");
   }
   else{
       printf("PORTCFG setting is succeeded\n");
       system("reboot");  //module must be reboot, the port configuration is
applied next power cycle
   }
  close(fd);
  return 0;
}
```

9. USB INTERFACE

LE910Cx includes a USB2.0 compliant Universal Serial Bus (USB) Transceiver, which operates at USB 2.0 High-speed (480Mbits/sec). By default, the module is configured as a USB peripheral mode.

The table below lists the available USB compositions:

Product ID	Description
1200	None mode
1201	DIAG + ADB + RMNET + NMEA + MODEM + MODEM + SAP
1203	RNDIS + DIAG + ADB + NMEA + MODEM + MODEM + SAP
1204	DIAG + ADB + MBIM + NMEA + MODEM + MODEM + SAP
1205	MBIM
1206	DIAG + ADB + ECM + NMEA + MODEM + MODEM + SAP
1250	RMNET + NMEA + MODEM + MODEM + SAP
1251	RNDIS + NMEA + MODEM + MODEM + SAP
1252	MBIM + NMEA + MODEM + MODEM + SAP
1253	ECM + NMEA + MODEM +MODEM + SAP
1254	MODEM + MODEM
1255	NMEA + MODEM + MODEM + SAP
1230	DIAG + ADB + RMNET + AUDIO + NMEA + MODEM + MODEM + SAP
1231	RNDIS + DIAG + ADB + AUDIO + NMEA + MODEM + MODEM + SAP
1260	DIAG + ADB + RMNET + NMEA + MODEM + MODEM + SAP
1261	DIAG + ADB + RMNET + NMEA + MODEM + MODEM + SAP
1262	DIAG + ADB + RMNET + NMEA + MODEM + MODEM + AUX

Table 7: LE910Cx USB Compositions

For more information, refer to #USBCFG command on AT commands Reference Guide.

9.1. Reading Current USB Product ID

Example:

```
include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
```

```
#include <fcntl.h>
#include <sys/wait.h>
#include <unistd.h>
#include <errno.h>

#define CURRENT_CONFIGURATION_FILE_NAME    "/data/usb/boot_hsusb_composition"
static int get_current_usb_configuration_name_id(char
*current_configuration_file_name);

static int get_current_usb_configuration_name_id(char
*current_configuration_file_name)
{
    char *composition_name_ptr = NULL;
    int composition_id = 0;
    char *linkname = NULL;
    ssize_t r = 0;

    if(current_configuration_file_name == NULL)
    {
        printf("current_configuration_file_name NULL pointer");
        return  -1;
    }

    linkname = malloc(PATH_MAX + 1);
    if (linkname == NULL) {
        printf("insufficient memory can't malloc\n");
        return  -1;
    }
    memset(linkname, 0, (PATH_MAX + 1));

   r = readlink(current_configuration_file_name,  linkname,  PATH_MAX);
   if (r < 0) {
        printf("readlink failed. r = %d\n", r);
        free(linkname);
        return  -1;
    }

   if (r < 4) {
       printf("File link error. r = %d\n", r);
       free(linkname);
       return  -1;
```

```
    }

   //Last 4 charachters on file path will be composition file name,
   //which are also the composition number and the information we're after.
   linkname[r] = '\0';

   composition_name_ptr = &linkname[r-4];
   printf("composition_name_ptr = %s\n", composition_name_ptr);

   composition_id = atoi(composition_name_ptr);
   printf("Current composition is %d\n",  composition_id);

   free(linkname);
   return composition_id;
}
int main(int argc, char *argv[])
{
     int pid = 0;
     pid =
get_current_usb_configuration_name_id(CURRENT_CONFIGURATION_FILE_NAME);
     printf("Current USB product ID = %d \n", pid);
     return 0;
}
```

9.2. Changing USB Composition

Example:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/wait.h>
#include <unistd.h>
#include <errno.h>

#define MAX_COMMAND_LEN                                  256
#define USB_COMPOSITION_SET_COMMAND            "usb_composition "
static int change_usb_composition(int  new_composition_name_id);
```

```
static int change_usb_composition(int  new_composition_name_id)
{
     char change_usb_composition_command[MAX_COMMAND_LEN] = {0};
     int res = 0;

     snprintf (change_usb_composition_command, MAX_COMMAND_LEN,  "%s  %d  n  y
n  n", USB_COMPOSITION_SET_COMMAND, new_composition_name_id);

     res = system(change_usb_composition_command);

     if(res == 0)
     {
        printf("USB composition was changed. Need to reboot.");
         return 0;
     }
     else
     {
         printf("Cannot execute %s command, returned value = %d",
change_usb_composition_command, res);
         return -1;
     }
}

int main(int argc, char *argv[])
{
      int pid = 0;
      int result = 0;
/*
      To change the USB composition to PID_1203 or the other PID, please set
to 1203 or the other PID as below.
*/
      pid = 1203;    // RNDIS + DIAG + ADB + NMEA + MODEM + MODEM + SAP
      result  = change_usb_composition(pid);
      if(result == 0)
      {
          system ("reboot");
      } else{
          printf("cannot change usb composition");
     }
      return 0;
}
```

10. EXCEPTION INFORMATION

10.1. Reading Exception Information

You can read the exception information from the below path:

/sys/class/misc/telit_rawdata/fatal_info

Example:

```
~ # cat /sys/class/misc/telit_raw_data/fatal_info
#EXCEPINFO:
1,"M0F.220006","2019/11/05","02:48:20",1704,"dsatm2mgen.c","Assertion 0
failed:PC 837E6EDC:LR 837E6454:SP 869FE0B8"
#EXCEPINFO: 2,"","","",0,"",""
#EXCEPINFO: 3,"","","",0,"",""
#EXCEPINFO: 4,"","","",0,"",""
#EXCEPINFO: 5,"","","",0,"",""
```

10.2. Clearing Stored Information

You can clear the stored exception information by writing '0' to below path:

`/sys/class/misc/telit_rawdata/fatal_info`

Example:

```
~ # cat /sys/class/misc/telit_raw_data/fatal_info
#EXCEPINFO:
1,"M0F.220006","2019/11/05","02:48:20",1704,"dsatm2mgen.c","Assertion 0
failed:PC 837E6EDC:LR 837E6454:SP 869FE0B8"
#EXCEPINFO: 2,"","","",0,"",""
#EXCEPINFO: 3,"","","",0,"",""
#EXCEPINFO: 4,"","","",0,"",""
#EXCEPINFO: 5,"","","",0,"",""
~ # echo 0 > /sys/class/misc/telit_raw_data/fatal_info
~ # cat /sys/class/misc/telit_raw_data/fatal_info
#EXCEPINFO: 1,"","","",0,"",""
#EXCEPINFO: 2,"","","",0,"",""
#EXCEPINFO: 3,"","","",0,"",""
#EXCEPINFO: 4,"","","",0,"",""
#EXCEPINFO: 5,"","","",0,"",""
```

11. WLAN INTERFACE

11.1. Setting WLAN SDIO Clock

You can set SDIO clock for the WLAN interface with write <clock> value to the below file. The changed value will be applied when the WLAN is started. If this value changes while the WLAN is already turned on, it must be restarted.

Note: The changed value by the user will be maintained even after module reboot or FW update.

```
/sys/class/misc/telit_raw_data/wlan_max_clock
```

The <clock> value is mapped as shown in the table below.

Value	Frequency
1	400khz
2	20Mhz
3	25Mhz
4	50Mhz
5	100Mhz
6 (default)	200Mhz

Table 8: WLAN SDIO Clock Value

For example, if you like to set the SDIO clock to 50Mhz,

```
~ # echo 4 > /sys/class/misc/telit_raw_data/wlan_max_clock
```

11.2. Getting Current WLAN SDIO Clock

To get the current and applied maximum SDIO clock for the WLAN interface, use the following file:

```
/sys/class/misc/telit_raw_data/wlan_max_clock
```

The currently configured <clock> value will be returned.

For example, If the 50Mhz has been configured,

```
~ # cat /sys/class/misc/telit_raw_data/wlan_max_clock
4
```

12. OPM INTERFACE

LE910Cx module provides an OPM (Operating Mode) interface to control the module's operating mode and the Power Saving Mode (PSM).

This interface allows you to change the behavior of your modem through a user application on Linux, which provides the same behavior and modes as the +CFUN command.

Note:

For more information on AT+CFUN command, refer to AT commands Reference Guide.

For more information on Power Saving Mode, refer to PSM Application Note for more details)

12.1. Using OPM Interface

The device driver node (/dev/telit_opm) can be used to control modem operating mode by the user application, and the parameters for each mode are as follows.

The table below lists the supported operating modes for the LE910Cx.

Operating Mode	Description
1	Mobile full functionality with power saving disabled
2	Disable TX (Not support)
4	Disable both TX and RX
5	Mobile full functionality with power saving enabled
6	Mobile reboot
7	Offline mode
8	FTM

Table 9: LE910Cx Supported Operating Modes

Example:

```
#define OPM_DEV_PATH         "/dev/telit_opm"

#define M2M_OPM_MAGIC        'o'
#define IOCTL_M2M_OPM_SET          _IOW( M2M_OPM_MAGIC, 0, unsigned int )
#define IOCTL_M2M_OPM_GET          _IOW( M2M_OPM_MAGIC, 1, unsigned int )

#define M2M_OPM_MODE_MAX     9
```

```
/* OPM command parameters */
enum
{
  M2M_OPM_CMD_SET_VAL = 0,
  M2M_OPM_CMD_GET_VAL,
  M2M_OPM_CMD_MAX
};

 int main(int argc, char *argv[])
{
  int dev = 0;
  int ret = 0;
  unsigned int opm_val=0;

  if(atoi(argv[1]) >= M2M_OPM_CMD_MAX)
  {
    perror("[OPM] cmd parameter out of range \n");
    return -1;
  }
  if(argc == 3)
  {
    opm_val = atoi(argv[2]);
}

  dev = open(OPM_DEV_PATH, O_RDWR);
  if(dev < 0)
  {
    perror("[OPM] driver open failed \n");
    return -1;
  }

  switch(atoi(argv[1]))
  {
    case M2M_OPM_CMD_SET_VAL:
      ret = ioctl(dev, IOCTL_M2M_OPM_SET, opm_val);
      break;

    case M2M_OPM_CMD_GET_VAL:
      ret = ioctl(dev, IOCTL_M2M_OPM_GET, &opm_val);
```

```
      if(ret)
        return -1;
      else
        return opm_val;
      break;

    default:
      break;
  }

  close(dev);
  return ret;
}
```

12.2. Configuring PSM DTR and WAKE_LOCK

The module can enter the power saving mode when all the below condition are met.

- USB disconnected
- UART's DTR off
- No WAKE_LOCK

To satisfy a PSM condition, users who are unable to control the DTR-pin on an external device can turn off a UART DTR by configuring it as a GPIO via AT#V24CFG (/dev/m2m drv cfg). For more information, see chapter 8 UART Interface.

In addition, if the user wants to maintain wake-up status after setting the operating-mode 5 (+CFUN=5) is set, the user application can use WAKE_LOCK to prevent the module from entering Sleep, as shown below.

Example:

```
// Set WAKE_LOCK for keeping wake-up
system("echo telit_opm > /sys/power/wake_lock")

// Set WAKE_UNLOCK for entering PSM
system("echo telit_opm > /sys/power/wake_unlock")
```

Warning: Users who use this driver should enter power saving mode only if they have the resources to wake up all the time.

"4-Disable RF" and "7-Offline" modes should be used with caution.

Otherwise, the module may not wake up from Sleep status.

(For more information refer to Software User Guide)

13. THERMAL SENSOR INTERFACE

The LE910Cx has a thermal sensor interface that allows a user application or a Linux shell to read the module's temperature.

There are six thermal sensors (five TSENS and one PA_THERM) on LE910Cx, two of which are used for thermal mitigation as listed below.

Sensor	Area
TSENS 3	MDM9207
PA_THERM	PA (Power Amp.)

Table 10: Thermal Mitigation Sensors

13.1. Reading Thermal Sensors

The sysfs node '/sys/class/thermal' can be used to read modem temperature by the user application or shell, and the nodes are as follows.

Sensor	Sysfs Node	Value
TSENS 3	/sys/class/thermal/thermal_zone3/temp	Degree Celsius
PA_THERM	/sys/class/thermal/thermal_zone5/temp	Degree Celsius

Table 11: Read Thermal Sensors

Example:

```
// Read TSENS 3

/ # cat /sys/class/thermal/thermal_zone3/type
tsens_tz_sensor3

/ # cat /sys/class/thermal/thermal_zone3/temp
32


// Read PA_THERM

/ # cat /sys/class/thermal/thermal_zone5/type
pa_therm0

/ # cat /sys/class/thermal/thermal_zone5/temp
28
```

14. ADC INTERFACE

LE910Cx provides an Analog-to-Digital Converter (ADC) interface that can be used to read data from a user application or a Linux shell.

14.1. Reading ADC Values

The sysfs node '/sys/devices/qpnp-vadc-8' can be used to read three ADC channels by the user application or shell, and the nodes are as follows.

ADC Channels	Sysfs Node	Value
ADC1	/sys/devices/qpnp-vadc-8/adc1	Microvolts (µV)
ADC2	/sys/devices/qpnp-vadc-8/adc2	Microvolts (µV)
ADC3	/sys/devices/qpnp-vadc-8/adc3	Microvolts (µV)

Table 12: ADC Values

Example:

```
// Read ADC1 (input: 7V)
/ # cat /sys/devices/qpnp-vadc-8/adc1
Result:716000 Raw:7d2e

// Read ADC2 (input: 6V)
/ # cat /sys/devices/qpnp-vadc-8/adc2
Result:607000 Raw:78d7

// Read ADC3  (input: 1.7V)
/ # cat /sys/devices/qpnp-vadc-8/adc3
Result:1709000 Raw:a4c5
```

Note: Only "Result" is valid for values returned by ADC nodes.

In case of "Raw", user application cannot use it because it contains internal setting values.

ADC interface is not available on LE910C1-SA, LE910C1-ST, and LE910C1-SV products. (Supported only through AT command).

15. PRODUCT AND SAFETY INFORMATION

15.1. Copyrights and Other Notices

SPECIFICATIONS ARE SUBJECT TO CHANGE WITHOUT NOTICE

Although reasonable efforts have been made to ensure the accuracy of this document, Telit assumes no liability resulting from any inaccuracies or omissions in this document, or from the use of the information contained herein. The information contained in this document has been carefully checked and is believed to be reliable. Telit reserves the right to make changes to any of the products described herein, to revise it and to make changes from time to time without any obligation to notify anyone of such revisions or changes. Telit does not assume any liability arising from 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.

This document may contain references or information about Telit's products (machines and programs), or services that are not announced in your country. Such references or information do not necessarily mean that Telit intends to announce such Telit products, programming, or services in your country.

15.1.1. Copyrights

This instruction manual and the Telit products described herein may include or describe Telit copyrighted material, such as computer programs stored in semiconductor memories or other media. The laws in Italy and in other countries reserve to 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 of Telit's or its licensors' copyrighted material contained herein or described in this instruction manual, shall not be copied, reproduced, distributed, merged, or modified in any way without the express written permission of the owner. Furthermore, the purchase of Telit products shall not be deemed to grant in any way, neither directly nor by implication, or estoppel, any license.

15.1.2. Computer Software Copyrights

Telit and the Third Party supplied Software (SW) products, described in this instruction manual may include Telit's and other Third Party's copyrighted computer programs stored in semiconductor memories or other media. The laws in Italy and in other countries reserve to Telit and other Third Party, SW exclusive rights for copyrighted computer programs, including – but not limited to - the exclusive right to copy or reproduce in any form the copyrighted products. Accordingly, any copyrighted computer programs contained in Telit's products described in this instruction manual shall not be copied (reverse engineered) or reproduced in any manner without the express written permission of the copyright owner, being Telit or the Third-Party software supplier. Furthermore, the purchase of Telit products shall not be deemed to grant either directly or by implication, estoppel, or in any other way, any license under the copyrights, patents, or patent applications of Telit or other Third Party supplied SW, except for the normal non-exclusive, royalty free license to use arising by operation of law in the sale of a product.

15.2. Usage and Disclosure Restrictions

15.2.1. License Agreements

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

15.2.2. Copyrighted Materials

The Software and the documentation are copyrighted materials. Making unauthorized copies is prohibited by the law. The software or the documentation shall not be reproduced, transmitted, transcribed, even partially, nor stored in a retrieval system, nor translated into any language or computer language, in any form or by any means, without prior written permission of Telit.

15.2.3. High Risk Materials

Components, units, or third-party goods used in the making of 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: operations 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 eligibility for such High-Risk Activities.

15.2.4. Trademarks

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

15.2.5. Third Party Rights

The software may include Third Party's software Rights. In this case the user agrees to comply with all terms and conditions imposed in respect of such separate software rights. In addition to Third Party Terms, the disclaimer of warranty and limitation of liability provisions in this License, shall apply to the Third-Party Rights software as well.

TELIT HEREBY DISCLAIMS ANY AND ALL WARRANTIES EXPRESSED OR IMPLIED FROM ANY THIRD PARTY REGARDING ANY SEPARATE FILES, ANY THIRD-PARTY MATERIALS INCLUDED IN THE SOFTWARE, ANY THIRD-PARTY MATERIALS FROM WHICH THE SOFTWARE IS DERIVED (COLLECTIVELY "OTHER CODES"), AND THE USE OF ANY OR ALL OTHER CODES 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 CODES MUST BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST OF 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 CODES 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.

15.2.6. Waiver of Liability

IN NO EVENT WILL TELIT AND ITS AFFILIATES BE LIABLE FOR AY DIRECT, INDIRECT, SPECIAL, GENERAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY INDIRECT DAMAGE OF ANY KIND WHATSOEVER, INCLUDING BUT NOT LIMITED TO REIMBURSEMENT OF COSTS, COMPENSATION OF ANY DAMAGE, LOSS OF PRODUCTION, LOSS OF PROFIT, LOSS OF USE, LOSS OF BUSINESS, LOSS OF DATA OR REVENUE, WHETHER OR NOT THE POSSIBILITY OF SUCH DAMAGES COULD HAVE BEEN REASONABLY FORESEEN, CONNECTED IN ANY WAY TO THE USE OF THE PRODUCT/S OR TO THE INFORMATION CONTAINED IN THE PRESENT DOCUMENTATION, EVEN IF TELIT AND/OR ITS AFFILIATES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES OR THEY ARE FORESEEABLE OR FOR CLAIMS BY ANY THIRD PARTY.

15.3. Safety Recommendations

Make sure the use of this product is allowed in your country and in the environment required. The use of this product may be dangerous and has to be avoided in areas where:

- it can interfere with other electronic devices, particularly in environments such as hospitals, airports, aircrafts, etc.
- there is a risk of explosion such as gasoline stations, oil refineries, etc. It is the responsibility of the user to enforce the country regulation and the specific environment regulation.

Do not disassemble the product; any mark of tampering will compromise the warranty validity. We recommend following the instructions of the hardware user guides for correct wiring of the product. The product has to be supplied with a stabilized voltage source and the wiring has to be conformed to the security and fire prevention regulations. The product has to be handled with care, avoiding any contact with the pins because electrostatic discharges may damage the product itself. Same cautions have to be taken for the SIM, checking carefully the instruction for its use. Do not insert or remove the SIM when the product is in power saving mode.

The system integrator is responsible for the functioning of the final product. Therefore, the external components of the module, as well as any project or installation issue, have to be handled with care. Any interference may cause the risk of disturbing the GSM network or external devices or having an impact on the security system. Should there be any doubt, please refer to the technical documentation and the regulations in force. Every module has to be equipped with a proper antenna with specific characteristics. The antenna has to be installed carefully in order to avoid any interference with other electronic devices and has to guarantee a minimum distance from the body (20 cm). In case this requirement cannot be satisfied, the system integrator has to assess the final product against the SAR regulation.

The equipment is intended to be installed in a restricted area location.

The equipment must be supplied by an external specific limited power source in compliance with the standard EN 62368-1.

The European Community provides some Directives for the electronic equipment introduced on the market. All of the relevant information is available on the European Community website:

https://ec.europa.eu/growth/sectors/electrical-engineering_en

16. GLOSSARY

HSIC	High-Speed Inter-Chip
I2C	Inter-Integrated Circuit
GPIO	General Purpose Input/Output
SDA	Serial Data Line
SCL	Serial Clock Line
SPI	Serial Peripheral Interface
USB	Universal Serial Bus
UART	Universal Asynchronous Receiver Transmitter

17. DOCUMENT HISTORY

Revision	Date	Changes
7	2022-03-14	Updated clauses: 4.1.2. Controlling Ethernet Interface in User Application 5.1. Using GPIO Interface 5.2. Using GPIO Interrupt
6	2022-02-11	Template updated Minor editorial changes
5	2021-06-09	Update: - Section 4.1.3. How to disable/enable the "CLK125" of external Marvell PHY (88E1512/5)
4	2021-06-04	New: - Section 4.1.3. How to disable/enable the "CLK125" of external Marvell PHY (88E1512/5) Update: - Section 5. GPIO INTERFACE
3	2020-12-23	New: - Section 5.2. How to use GPIO interrupt Update: - Section 13. THERMAL SENEOR INTERFACE - Section 5.1. How to use GPIO interface - Section 9. USB INTERFACE
2	2020-11-04	New: - Section 13. THERMAL SENEOR INTERFACE - Section 14. ADC INTERFACE Update: - Section 4. ETHERNET INTERFACE
1	2020-01-21	New: - Section 11. WLAN INTERFACE - Section 12. OPM INTERFACE Update: - Applicability table
0	2019-11-22	Initial version

From Mod.0809 rev.3

Connect to our site and contact our technical support team for any question

www.telit.com