EZ863H
Software user guide
Rev.1 – 19/03/13

Contents

1. Introduction

1.1 Overview

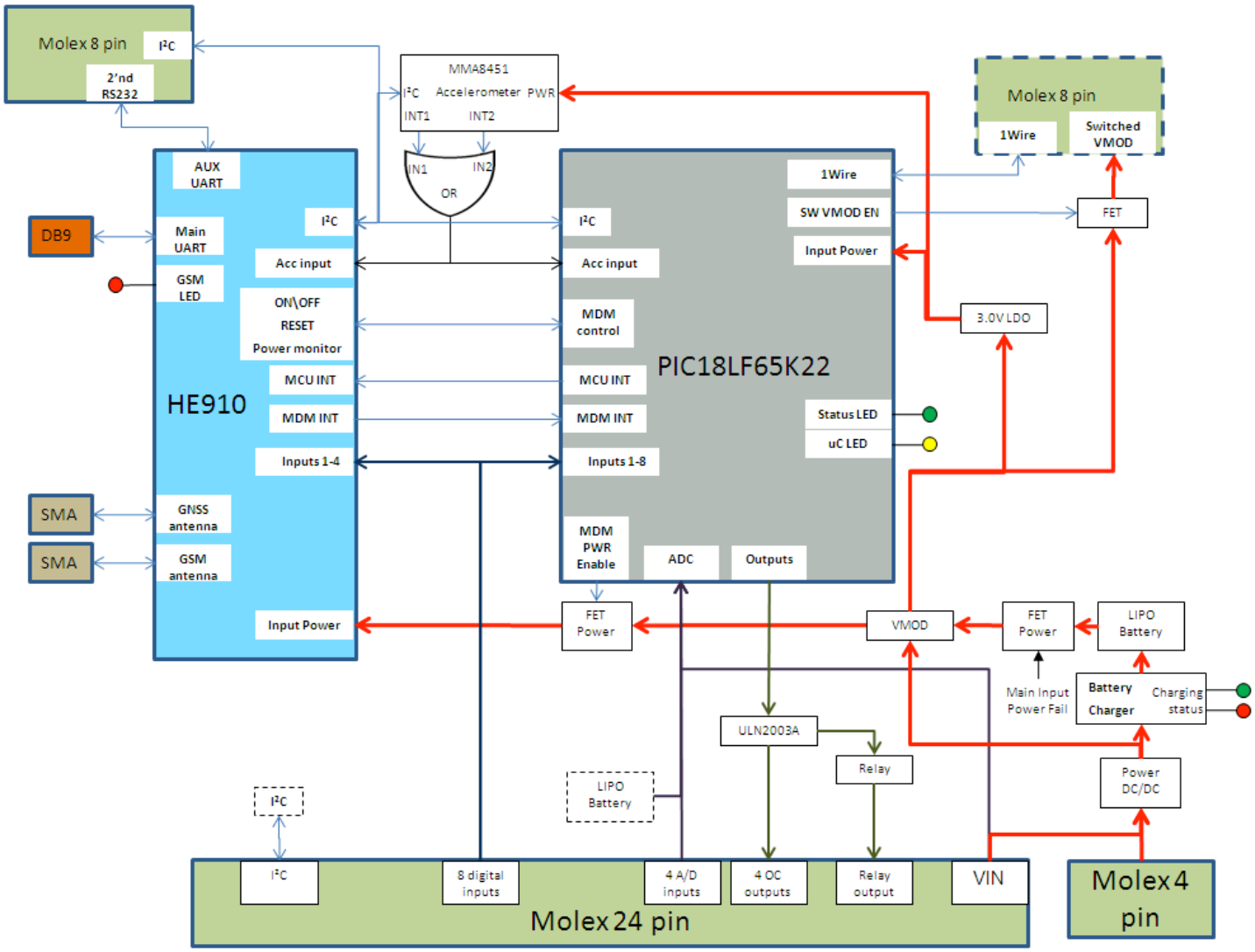
Aim of this document is to describe how to use the Python script in order to work with the features, functions and interfaces of the EZ863H unit.
The EZ863H is a complete system solution for cellular based product such AVL GSM GPS applications, metering, data capture, etc…
The unit based on Telit xE910 series.
The unit includes Telit cellular engine with Python, 3 Axis accelerometer sensors, microcontroller for modem watchdog and peripherals support and LiPo battery with charger.

1.2 Related documents

The following documents are related to this user guide:

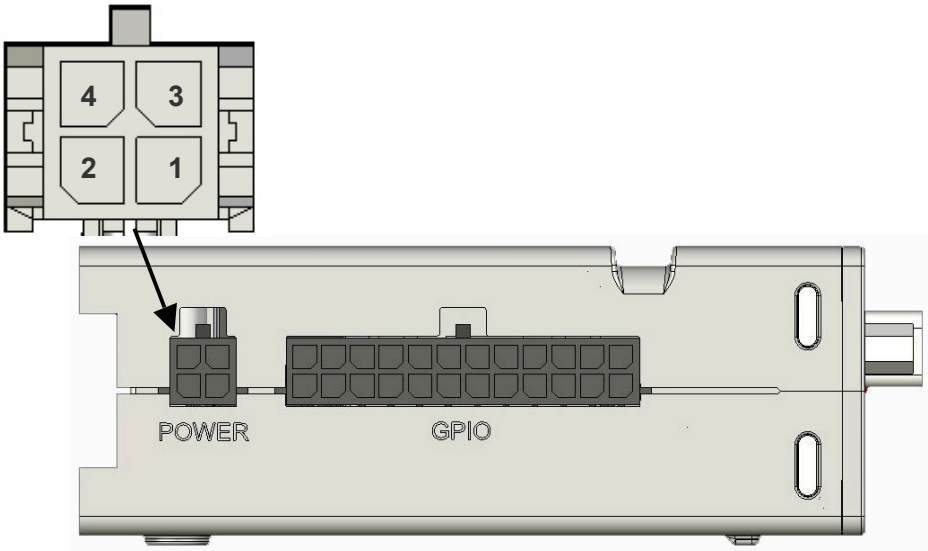
- [1] MMA8451Q
- [2] Telit_HE910_Easy_Script_in_Python_r0
- [3] EZ863H_Product_Description
- [4] EZ863H_3G_Product_Description
- [5] Telit_HE910_Family__Ports_Arrangements_r4

2. EZ863H hardware block diagram



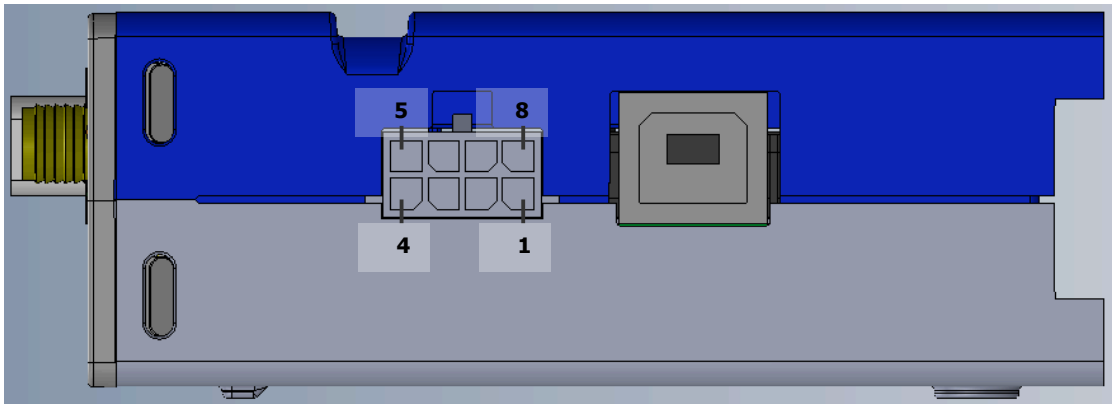
3. Interface Descriptions

3.1 Molex 4 pin connector – Power connector



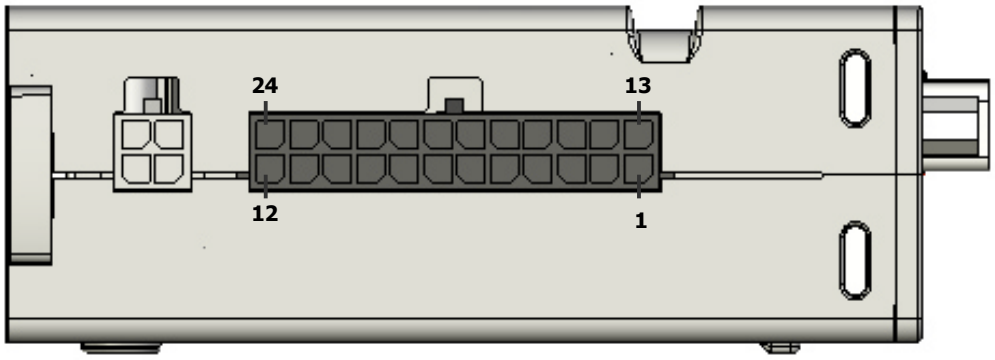
Pin	Signal name	I/O	Description
1	Power	PWR	Power input
2	Digital input 1	I	0-50V digital input (same signal as in 24 pin connector)
3	GND	PWR	Power ground
4	Digital input 2	I	0-50V digital input (same signal as in 24 pin connector)

3.2 Molex 8 pin connector – AUX interface



Pin	Signal name	Description
1	AUX RX	Modem AUX port RX – RS232 level
2	AUX TX	Modem AUX port TX – RS232 level
3	I2C ACL	I2C bus clock
4	I2C SDA	I2C bus data
5	SW VMOD	Switched VMOD
6	-----	
7	1Wire	1Wire bus
8	GND	

3.3 Molex 24 pin connector – IO interface



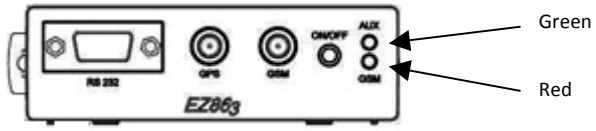
Pin	Signal name	I/O	Description
1	Digital input 1	I	0-50V digital input
2	Digital input 2	I	0-50V digital input
3	Digital input 3	I	0-50V digital input – internal 100K pull-up
4	Digital input 4	I	0-50V digital input – internal 100K pull-up
5	OC1	O	Open collector 500mA output
6	OC2	O	Open collector 500mA output
7	OC3	O	Open collector 500mA output
8	OC4	O	Open collector 500mA output
9	ADC1	I	0-50 12bit analog input
10	ADC2	I	0-50 12bit analog input
11	VMOD	O	VMOD voltage output for external accessory activation
12	DGND	PWR	Digital ground
13	Relay A	O	Relay output A
14	Relay B	O	Relay output B
15	Digital input 5	I	0-50V digital input
16	Digital input 6	I	0-50V digital input
17	Digital input 7	I	0-50V digital input – internal 100K pull-up
18	Digital input 8	I	0-50V digital input – internal 100K pull-up
19	ADC3	I	0-50 12bit analog input
20	ADC4	I	0-50 12bit analog input
21	I2C SCL		I2C bus clock
22	I2C SDA		I2C bus data
23	PLG GND	PWR	Plug ground
24	Vin	PWR	Input voltage

3.4 Connectivity map

Signal name	I/O	Description
Digital input 1	I	Modem – GPIO7, uC – HVIO1
Digital input 2	I	Modem – GPIO8, uC – HVIO2
Digital input 3	I	Modem – GPIO9, uC – HVIO3
Digital input 4	I	Modem – GPIO10, uC – HVIO4
OC1	O	uC – OC1
OC2	O	uC – OC2
OC3	O	uC – OC3
OC4	O	uC – OC4
ADC1	I	uC – ADC1
ADC2	I	uC – ADC2
Relay	O	uC – Relay
Digital input 5	I	uC – HVIO5
Digital input 6	I	uC – HVIO6
Digital input 7	I	uC – HVIO7
Digital input 8	I	uC – HVIO8
ADC3	I	uC – ADC3
ADC4	I	uC – ADC4
I2C SCL	Bus	Modem – GPIO2
I2C SDA	Bus	Modem – GPIO3
Front panel red LED	O	Modem – GPIO1
Front panel green LED	O	uC – Green LED
uC WD signal	uC-I, modem-O	Reset Python watchdog, Modem – GPIO4
Wake-up event signal	uC-O, modem-I	Event report from the uC to the modem, Modem – GPIO6
ACC INT	I	Accelerometer interrupts OR, Modem – GPIO5

3.4 Status LEDs

3.4.1 Front panel LEDs



3.4.1.1 Red LED

The Red LED is connected to GPIO1, OFF by default.
The LED can use for Network status or controlled by the user using the modem's GPIO1.
To activate GSM status Red LED: "AT#GPIO=1,0,2; #SLED=2,1,1"

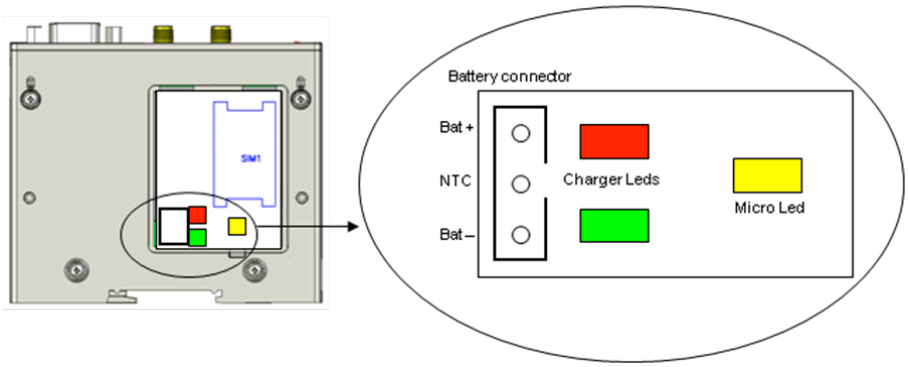
Red LED status	Device Status
Slow interrupt sequence (Ton 0,3s, Toff 2.7s)	Registered full service
Medium interrupt sequence (Ton 0,5s, Toff 0.5s)	Net search / Not registered
Fast interrupt sequence (Ton 0,1s, Toff 0.1s)	Fail to connect to server
Permanently off	device off

Red LED ON: "AT#GPIO=1,1,1"
Red LED OFF: "AT#GPIO=1,0,1"

3.4.1.2 Green LED

The green LED is connected to micro controller.
The LED can be controlled by the user using the I2C bus.

3.4.2 SIM house LEDs



3.4.2.1 Yellow LED

The yellow LED controlled by the micro controller to indicate its status
In order to save energy, it's possible to disable this LED over the I2C bus

3.4.2.2 Charger LEDs

The red and the green LEDs are indicates the charger status

4. Python I2C communication

The Python script is communicating with the modem and the accelerometer over the I2C but.
The I2C bus is connected over GPIOs 2 and 3 while SCL=GPIO2 and SDA=GPIO3.
In order to configure the I2C bus in Python script, GtIIC library must be used with the **readwrite** function.

4.1 Accelerometer

The accelerometer uses in the EZ863H unit is MMA7451, the sensor can send the acceleration data or detect changes in space.
The accelerometer has 2 programmable interrupt pins for 7 interrupt sources that are router via OR gate to GPIO5:

1. Freefall or Motion Detection: 1 channel (INT1 or 2)
2. Pulse Detection: 1 channel (INT1 or 2)
3. Jolt Detection: 1 channel (INT1 or 2)
4. Orientation (Portrait/Landscape) detection with programmable hysteresis

INT1 and INT2 of the accelerometer are normally high and low when interrupt detected, once the Python script start working, it must set the INT1 and INT2 to work as normally low and high when interrupt detected on the accelerometer register 0x2C

4.2 PIC microcontroller

The PIC microcontroller has the following functionalities:

1. Use as hardware watchdog for the Python script
2. Monitoring the battery
3. Wake-up the modem (and therefore the Python script) from shut down when external event occurred
4. Measure the input voltage and the battery voltage
5. Measure the 4 analog inputs with capability to use digital filter that emulate capacitor between 100ms to 2550ms
6. Extent the digital inputs to 8
7. Activate and deactivate the outputs (OCs and relay)
8. Communicate with iButton over 1-wire bus
9. Communicate with up to 4 temperature sensors over 1-wire bus

4.3 PIC uC

In order to initialize the I2C class with the uC address, the GtIIC library needs to be use.
The uC address is 0x04.

Example:

```
I2C_SCL = 2 # GPIO used for SCL pin
I2C_SDA = 3 # GPIO used for SDA pin

I2C_ADDR = 0x04 # myIIC1 address
IIC_MCU = GtIIC.new(I2C_SDA, I2C_SCL, I2C_ADDR)
status = IIC_MCU.init()
```

4.3.1 Reading the digital inputs

In order to read the current snapshot of the digital inputs, resister 0x00 needs to be read

Address	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x00	HVIO Status	HVIO8	HVIO7	HVIO6	HVIO5	HVIO4	HVIO3	HVIO2	HVIO1

Example:

```
l_res = IIC_MCU.readwrite(chr(0x00), 1)
l_res = ord(l_res)
if(l_res & 0x01):
   print 'Input 1 high'
if(l_res & 0x02):
   print 'Input 2 high'
if(l_res & 0x04):
   print 'Input 3 high'
if(l_res & 0x08):
   print 'Input 4 high'
if(l_res & 0x10):
   print 'Input 5 high'
if(l_res & 0x20):
   print 'Input 6 high'
if(l_res & 0x40):
   print 'Input 7 high'
if(l_res & 0x80):
   print 'Input 8 high'
```

4.3.2 Activate and deactivate the outputs

In order to activate or deactivate an output, resister 0x16 bit needs to be set or reset

Address	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x16	I/O CTRL 0	Charger En	RS232_En	MCULED_En	RELAY	OC4	OC3	OC2	OC1

1. Bits 0-3 – OC1-OC4: controls the open collector 1-4 (OC1-4) outputs.
   Set this bit to 1 in order to activate the correspond output
   Set this bit to 0 in order to deactivate the correspond output
2. Bits 4 – Relay: controls the relay output.
   Set this bit to 1 in order to activate the relay
   Set this bit to 0 in order to deactivate the relay
3. Bits 5 – MCULED_En: controls the yellow uC LED placed in the SIM housing.
   Set this bit to 1 in order to enable the uC status LED.
   Set this bit to 0 in order to disable the uC status LED and save energy.
4. Bits 6 – RS232_En: controls the RS232 level shifters.
   Set this bit to 1 in order to enable the RS232 level shifters.
   Set this bit to 0 in order to disable the RS232 level shifters, please note that the modem cannot send any data while the level shifters are disabled.
5. Bit 7 - Charger En: enable or disable the LiPo charger.
   Set this bit to 1 in order to enable the LiPo battery charging.
   Set this bit to 0 in order to save energy.

Example:

```
l_res = IIC_MCU.readwrite(chr(0x16), 1)
l_res = ord(l_res)
#turn the relay on
l_res = l_res | 0x10
IIC_MCU.readwrite(chr(0x16)+chr(l_res), 0)
#turn OC1 on
l_res = l_res | 0x01
IIC_MCU.readwrite(chr(0x16)+chr(l_res), 0)
#turn the relay off
l_res = l_res & 0xEF  #0xEF=not(0x10)
IIC_MCU.readwrite(chr(0x16)+chr(l_res), 0)
```

4.3.3 Watchdog

The watchdog has 3 states:

1. Watchdog disabled
2. Watchdog enabled and watching the Python script
3. Watchdog suspended while the modem is turned off and the Python script is not running

Watchdog registers:

Address	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x03	WD timeout	0 (WDT Disabled) OR (10 - 255 Sec)							
0x14	CONFIG BIT's 0						WD_SHDN		

In the unit first turn on, the watchdog is disabled; bit #2 on register 0x14 is set to 0 and resister 0x03 is 0x00.
When the Python script start running it should set the watchdog timeout (must be >10) to register 0x03 and the watchdog will start to work.
Example - Enable the watchdog with one minute timeout:
l_res = IIC_MCU.readwrite(chr(0x03) + chr(60), 0)

Keeping the watchdog alive is done with toggling GPIO4 on and off

Once the watchdog is enabled it been saved in the NVM (non volatile memory) and therefore will not be turn off unless register 0x03 will be set to 0.
By default the watchdog timer is set to 4 minutes, if the modem is turned on and the watchdog is enabled and the modem will not toggle GPIO4 within 4 minutes, the MCU will restart the modem over and over.

Keeping the watchdog alive – Python example (first GPIO10 needs to be set as output):
GPIO.setIOvalue(4, 0)
GPIO.setIOvalue(4, 1)

When the Python\modem needs to be shut down in order to put the unit in low power mode, the watchdog needs to be suspended.
In suspend mode, the MCU ignore the watchdog as long as the modem is turned off, once the modem is turned back on, the watchdog timer is set to 10 minutes, the watchdog suspend bit (bit2 in 0x14) is cleared and the MCU waits for GPIO4 toggling.
This way the MCU will reset the modem until the Python script will start running and toggle GPIO4.

The default watchdog timer is 4 minutes, in order to change this value, 0x04 register must be set. The watchdog timer is not been saved in the NVM, so it needs to be set every time the Python started.
If the watchdog time till be set to 0 minutes, the MCU will set it to 1 minute.

4.3.4 Input voltage measurement

The MCU measure the input voltage and stores the two bytes result in 0x04 and 0x05 registers, while 0x05 is the MSB byte and 0x04 is the LSB byte.
The result scale is mV.

Address	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x04	ADC Vin (mV)	LSB							
0x05		MSB							

Read the input voltage – Python example:
l_res = IIC_UC.readwrite(chr(0x04), 2) #Read the two bytes input voltage
SER.send('Input voltage=%d\r\n' % (ord(l_res[0]) + (ord(l_res[1])*256)))

4.3.5 Internal battery voltage measurement

The MCU measure the battery voltage and stores the two bytes result in 0x06 and 0x07 registers, while 0x07 is the MSB byte and 0x06 is the LSB byte.
The result scale is mV.

Address	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x06	ADC Vbat (mV)	LSB							
0x07		MSB							

Read the battery voltage – Python example:

```
l_res = IIC_UC.readwrite(chr(0x06), 2)   #Read the two bytes battery voltage
SER.send('Battery voltage=%d\r\n' % ( ord(l_res[0]) + (ord(l_res[1])*256) ))
```

4.3.6 ADC1-4 measurement

The MCU measure the ADC1-4 inputs and stores the result in series of two bytes from 0x08 to 0x0F registers, while the first byte is the LSB byte.
The result scale is mV.

Address	Description	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x08	ADC 1	LSB							
0x09		MSB							
0x0A	ADC 2	LSB							
0x0B		MSB							
0x0C	ADC 3	LSB							
0x0D		MSB							
0x0E	ADC 4	LSB							
0x0F		MSB							