G3 LUA API (fw_G3_4_4_x)

LUA interpreter version: 5.1.5

Revision log:

V2.5
>CANbus store to buffer
V2.4
>add zigbee_send function
V2.3
>update on Modbus master examples
V2.2
>add support for Zigbee (via usb port)
>add support for multi serial port (G3 extended ports version)
>add support for timeout in serial port & sms send/receive functions
V2.1
>add support for CANbus (CAN2.0A & 2.0B)
V2.0
>add support for GPS (standalone mode, non-assist)
>add support for serial port handle hex/bin data
>add support for standard Modbus master (Modbus/RTU & Modbus/TCP)
>add support for special Modbus protocol using 32-bits registers (available on custom G3 firmware)

CONTENTS Page

A. Introduction to LUA

Lua is a powerful scripting language used in many embedded devices worldwide.

www.lua.org

The Lua script interpreter is embedded into the Linux/OpenWRT operating system of the FATBOX G3.

With Lua script running, user can execute customized functions to

- Monitor digital input to create alarms/alerts.
- Manage the sending & receiving of SMS messages.
- Save events/data to a file in flash memory.
- Handle Serial RS232/485 communications with serial devices.
- Handle TCP/UDP communications with host or client.

First Lua program

1. Open text editor that can set EOL as Unix/Linux format (eg Notepad++).
2. Enter the following:

```
require "G3"
green_tick_led=53
while true do
        switch_led(green_tick_led,1)
        sleep(0.2)
        switch_led(green_tick_led,0)
        sleep(0.2)
end
```

3. Save the file as 'user.lua' to a USB drive with label 'FATBOX'.
   Then create a folder '/user' and move the file inside this folder.
   Confirm the file properties is type .lua
4. Connect the USB drive to the FATBOX G3's USB port.
5. Login via web browser using default ETH0 port IP 192.168.1.1
6. Click 'Download to FATBOX' under the LUA Script Management.
7. Click 'Execute user.lua Script' to run the script.
8. The blinking LED on the FATBOX G3 confirms the script running.

Run Lua from console

1. Connect console via ssh. Login as 'root'.
2. Enter command 'lua /user/user.lua'

B. Cellular Module functions

Reboot G3 router

reboot()
soft reboot the G3 router
Example:

```
require “G3”
if read_IN() then
        reboot()
end
```

Check cellular data status

cellular_status()
returns 0 if not connected to cellular data
returns 1 if connected to cellular data
Example:

```
require "G3"
if cellular_status() then
        print ("3G/4G data connection up")
end
```

Check cellular signal strength

check_signal(timeout_sec)
returns 0 when there is no +CSQ signal received
returns +CSQ:[n] number between 0 and 31 indicating the signal strength level
Example:

```
require “G3”
while true do
        csq=check_signal(10)                --timeout=10sec
        print(csq)
        if csq ~= 0 then
                rssi=tonumber(string.match(csq, "%d+"))
                print(rssi)
        else rssi=0 end
        sleep(1)
end
```

Switch ON/OFF cellular data

cellular_on()
cellular_off()
switch on/off the cellular data connection.
Example:

```
require "G3"
if read_IN() then
        cellular_on()
else
        cellular_off()
end
```

Pause LUA program

sleep(x)
Sleeps for x seconds
Example

```
require "G3"
sleep(0.5)
sleep(3)
```

Ping function

ping(host,interface)
returns 0 if ping failed
returns 1 if ping success
host=valid ip address or domain name eg "8.8.8.8" or "www.lua.org"
interface="eth0","eth1" or omitted for default route
Example

```
require "G3"
pingcam1=ping("192.168.1.100","eth0")
pingcam2=ping("10.1.1.200","eth1")
pingdnsserver=ping("8.8.8.8")
pingdomain=ping("www.microsoft.com")
```

C. I/O functions

Read position of DIP switch

read_dip(pos)
returns the DIP switch position for program selection/user feature activation.

'pos' value	DIP switch	Returns
1	1	1=down, 0=up
2	2	1=down, 0=up
3	3	1=down, 0=up
4	4	1=down, 0=up

Example:

```
require "G3"
if read_dip(2) then
        print ("dip switch #2 read position down!")
else
        print ("dip switch #2 read position up!")
end
```

Read digital input

read_IN()
returns 0 when IN terminal is open circuit.
returns 1 when IN terminal is connected to GND terminal.

Example:

```
require "G3"
if read_IN() then
        print ("IN terminal connected to GND!")
else
        print ("IN terminal is open")
end
```

Trigger LED output

switch_led(n,level)
switch on or off the on-board LEDs

note: By default, these LEDs are controlled by system to indicate signal strength and data connection status.

'n' value	LED	'level' value
53	Tick sign (GRN)	0=off, 1=on
54	Cross sign (RED)	0=off, 1=on
51	Signal low (GRN)	0=off, 1=on
52	Signal high (GRN)	0=off, 1=on

Example:

```
require "G3"
while true do
        switch_led(53,1)
        sleep(0.2)
        switch_led(53,0)
        sleep(0.2)
end
```

Blink LED output

blink_led(n, time)
blinks the on-board LEDs for a specific time period

note: By default, these LEDs are controlled by system to indicate signal strength and data connection status.

'n' value	LED	'time' value
53	Tick sign (GRN)	sec
54	Cross sign (RED)	sec
51	Signal low (GRN)	sec
52	Signal high (GRN)	sec

Example:

```
require "G3"
while true do
        blink_led(53,0.2)
        sleep(0.2)
        blink_led(54,0.2)
        sleep(0.2)
        blink_led(51,0.2)
        sleep(0.2)
        blink_led(52,0.2)
end
```

D. SMS functions

Initialize cellular module for SMS text/pdu

init_modem(mode)
mode= “sms” or “pdu”
Example:

```
require “G3”
init_modem(“sms”)          --for sending SMS in text mode.
init_modem(“pdu”)          --for sending SMS in pdu mode.
```

Sending SMS as normal text

sms_send(number,message,timeout_sec)
number= mobile number in international format
message= standard text limited to 160 characters (GSM 7-bit alphabet)
Example:

```
require “G3”
init_modem(“sms”)
number=”+6xxxxxxxxxx”
message=”ALARM!!! Battery voltage low.”
while true do
        sleep(10)
        print (“DIGITAL IN = ”..read_IN())
        if (read_IN()==1) then
                print (“External IN triggered. ALARM sent!”)
                sms_send(number,message,10)                    --timeout=10sec
        end
end
```

Reading SMS as normal text

sms_read(timeout_sec)
returns 0 when there is no sms received
returns the sms message when sms is received
Example:

```
require  “G3”
init_modem(“sms”)
repeat
        sleep(10)
        print (“Checking for new sms…”)
        msg=sms_read(10)                --timeout=10sec
until msg ~= 0
print (msg)
```

Sending SMS in PDU mode

pdu_send(length, pdu, timeout_sec)
length = Total length in octets (8bits eg. FF) excluding SMSC data.
pdu = The actual PDU data in hex form.
Example:

```
length = "20"
--20 >Length (decimal) of PDU data in octets (exclude SMSC info)
pdu = "0011000B911689674523F10008FF0677ED6D88606F"
--00 >SMSC, zero means use SMSC stored in phone/SIM
--11 >SMS-SUBMIT
--00 >TP-Message-Reference
--0B >Length of phone number (11)
--91 >International format for phone number
--1689674523F1 >phone number=61987654321
--00 >TP-PID
--08 >TP-DCS Data Coding Scheme 00=ASCII 04=Binary 08=UCS2(Unicode)
--FF >TP-Validity-Period
--06 >TP-User-Data-Length
--77ED6D88606F >sms message in Chinese for "Short Message"
require “G3”
init_modem(“pdu”)
while true do
        sleep(10)
        print ("DIGITAL IN= "..read_IN())
        if (read_IN()==1) then
                print ("External IN triggered. PDU data sent!")
                pdu_send(length,pdu,10)          --timeout=10sec
        end
end
```

Reading SMS in PDU mode

pdu_read(timeout_sec)
returns 0 when there is no PDU message received
returns the PDU message when received
Example:

```
require  “G3”
init_modem(“pdu”)
repeat
        sleep(10)
        print (“Checking for new sms…”)
        msg=pdu_read(10)                    --timeout=10sec
until msg ~= 0
print (msg)
```

E. SERIAL PORT functions

Open/close serial RS232/485

serial_open(port,baud_rate,data_bits,stop_bits,parity)
open serial port with defined port number, baud rate, data bits, stop bits and parity.

serial_close(port)
close serial port with defined port number

serial_raw(port)
set serial port with defined port number to raw mode

Port number	Physical label name	Connector	Type	Signal description	Information
1	SERIAL	Screw terminal 4-way	RS232 RS485	TX,RX,GND D-,D+,GND	RS232/485 type select via G3 web config page.
2	SERIAL2	D-SUB9 (F)	RS232	TX(pin2),RX(pin3),GND(pin5)	G3 extended ports version.
-	CONSOLE^	D-SUB9 (F)	RS232	TX(pin2),RX(pin3),GND(pin5)	G3 extended ports version.

baud_rate = 2400, 4800, 9600, 19200, 38400, 57600, 115200
data_bits = 7, 8
stop_bits = 1, 2
parity = “NONE”, “EVEN”, “ODD”

Note^ : CONSOLE port is Linux console and not available for programming access.

Sending serial RS232/485

serial_send(port,message,timeout_sec)

Send a message in ASCII code
Example:

```
require "G3"
serial_open(1,115200,8,1,”NONE”)          --opens port1 @115200baudrate,8databit,1stopbit,no parity
message="SEND MESSAGE FROM LUA\r\n"
serial_send(1,message,10)                 --send message via port1 with timeout=10sec
print("send="..message)
serial_close(1)                           --close port1
```

Send a message in HEX/BIN value.
Example:

```
require “G3”
serial_raw(1)                             --set port1 to raw mode
serial_open(1,115200,8,1,”NONE”)          --opens port1 @115200baudrate,8databit,1stopbit,no parity
hexdata=string.char(0x01,0x04,0x1A,0x2B,0x3F)
serial_send(1,hexdata,10)                 --send hexdata via port1 with timeout=10sec
serial_close(1)                           --close port1
```

Reading serial RS232/485

serial_read(port,end_char,timeout_sec)
end_char = return when receive the end of message character
returns nil when there is no serial data received
returns serial data upon receiving end_char or upon timeout

Example (ASCII code):

```
require "G3"
serial_open(1,115200,8,1,"NONE")      --opens port1 @115200baudrate,8databit,1stopbit,no parity
end_char="\n"                         --return on receiving newline char
rxstr=serial_read(1,end_char,10)      --read port1, returns only upon end_char or timeout=10sec
print("serial read data = "..rxstr)
serial_close(1)                       --close port1
```

Example (HEX/BIN code):

```
require "G3"
serial_open(1,115200,8,1,"NONE")      --opens port1 @115200baudrate,8databit,1stopbit,no parity
end_char=string.char(0x00)            --return on receiving char value 0x00
rxstr=serial_read(1,end_char,10)      --read port1, returns only upon end_char or timeout=10sec
byte2 = string.byte(rxstr,2)          --convert non-printable char to numerical code
byte3 = string.byte(rxstr,3)
print(byte2, byte3)
serial_close(1)                       --close port1
```

serial_receive(port,length,timeout_sec)
length = serial input buffer maximum size
returns nil when there is no serial data received
returns serial data upon input buffer full or upon timeout

Example (ASCII code):

```
require "G3"
serial_open(1,115200,8,1,"NONE")      --opens port1 @115200baudrate,8databit,1stopbit,no parity
length=10                             --input buffer max size
rxstr=serial_receive(1,length,10)     --read port1, returns only upon input buffer full or timeout=10sec
print("serial receive data = "..rxstr)
serial_close(1)                       --close port1
```

Example (HEX/BIN code):

```
require "G3"
serial_open(1,115200,8,1,"NONE")      --opens port1 @115200baudrate,8databit,1stopbit,no parity
length=10                             -- input buffer max size
rxstr=serial_receive(1,length,10)     --read port1, returns only upon input buffer full or timeout=10sec
byte3 = string.byte(rxstr,3)          --convert non-printable char to numerical code
byte4 = string.byte(rxstr,4)
print(byte3, byte4)
serial_close(1)                       --close port1
```

F. Modbus master functions

Initialize Modbus master module

init_mbm()
Function to initialize Lua-modbus library into userspace.
This must be executed once before any Modbus function call in Lua script.

Open new connection context to Modbus slave

L1=mbm.open("rtu", baudrate, parity, data, stop, timeout)
returns a valid context L1 (>=0)

"rtu"	= **Modbus/RTU** device connected via SERIAL port RS232/RS485
baudrate	= serial baudrate of RTU device (1200,2400,4800,9600,19200,38400,57600,115200)
parity	= serial parity of RTU device ("N","E","O")
data	= serial data bit of RTU device (7,8)
stop	= serial stop bit of RTU device (1,2)
timeout	= response timeout (seconds)

L1=mbm.open("tcp", ip_address, ip_port, timeout)
returns a valid context L1 (>=0)

"tcp"	= **Modbus/TCP** device connected via ETH port ETH0/ETH1
ip_address	= IP address of Modbus device (eg "192.168.1.100")
ip_port	= IP port of Modbus device (default 502)
timeout	= response timeout (seconds)

Connect using connection context to Modbus slave

mbm.connect(L1)
L1 = a valid connection context (>=0) from mbm.open() function
returns 0 if connection successful
returns -1 if connection failed/timeout

Read Boolean status from Modbus slave

FC=01 for read discrete coils
data_8bitTable, status = mbm.fc1(L1, node, coil_address, coil_count, timeout)

FC=02 for read discrete inputs
data_8bitTable, status = mbm.fc2(L1, node, input_address, input_count, timeout)

Write Boolean state to Modbus slave

FC=05 for write single discrete coil
status = mbm.fc5(L1, node, coil_address, coil_state, timeout)

FC=15 for write multiple coils
status = mbm.fc15(L1, node, coil_address, coil_count, timeout, data_8bitTable)

Read data registers from Modbus slave

FC=03 for read holding registers (40,001 in old Modicon convention)
data_16bitTable, status = mbm.fc3(L1, node, reg_address, reg_count, timeout)

FC=04 for read input registers (30,001 in old Modicon convention)
data_16bitTable, status = mbm.fc4(L1, node, reg_address, reg_count, timeout)

Write data registers to Modbus slave

FC=06 for write single register
status = mbm.fc6(L1, node, reg_address, data_16bit, timeout)

FC=16 for write multiple registers
status = mbm.fc16(L1, node, reg_address, reg_count, timeout, data_16bitTable)

xxx_address	= address of first coil/input/register to be read/write
xxx_count	= number of coils/inputs/registers to be read/write
coilstate	= integer with value 0 or 1
data_16bit	= 16bit unsigned integer
data_8bitTable	= array of 8bit unsigned integer elements
data_16bitTable	= array of 16bit unsigned integer elements
L1	= a valid connection context for the Modbus device
node	= Modbus/RTU slave address
	= Modbus/TCP node = 1
timeout	= response timeout (seconds)
status	= returns 1 for read/write success

Turn on/off the debug messages

mbm.debug(L1,debug)

debug	= 0	(turn off)
	= 1	(turn on)

Note: Debug messages when running script in console only.

Disconnect from Modbus slave using connection context

mbm.disconnect(L1)
Disconnect the Modbus connection context L1.

Close/free the Modbus connection context

mbm.close(L1)
Close and free the Modbus connection context L1.

Example for Modbus/RTU:

```
require “G3”

init_mbm()

baud=115200
parity=”N”
data=8
stop=1
timeout=10

L1 = mbm.open(“rtu”,baud,parity,data,stop,timeout)
status1 = mbm.connect(L1)
mbm.debug(L1,1)

if status1==0 then

        print("mbm.connect test connect success")
        node=3
        reg_addr=2000
        reg_cnt=10
        datawrite_16bitTable={0x1111,0x2222,0x3333,0x4444,0x5555,0x6666,0x7777,0x8888,0x9999,0xAAAA}

        stat1 = mbm.fc16(L1,node,reg_addr,reg_cnt,timeout,datawrite_16bitTable)
        if stat1==1 then
                print("mbm.fc16 test write register success”)
        end

        dataread_16bitTable,stat2 = mbm.fc3(L1,node,reg_addr,reg_cnt,timeout)
        if stat2==1 then
                print ("mbm.fc3 test read register success")
                i=0
                while (i<reg_cnt) do
                        i=i+1
                        print(dataread_16bitTable[i])
                end
        end

        mbm.disconnect(L1)
end
mbm.close(L1)
```

Example for Modbus/TCP:

```
require “G3”

init_mbm()

ip_addr=”192.168.1.100”
ip_port=502
timeout=10

L1 = mbm.open(“tcp”,ip_addr,ip_port,timeout)
status1 = mbm.connect(L1)
mbm.debug(L1,1)

if status1==0 then

        print("mbm.connect test connect success")
        node=1
        reg_addr=1000
        reg_cnt=10
        datawrite_16bitTable={0x1111,0x2222,0x3333,0x4444,0x5555,0x6666,0x7777,0x8888,0x9999,0xAAAA}

        stat1 = mbm.fc16(L1,node,reg_addr,reg_cnt,timeout,datawrite_16bitTable)
        if stat1==1 then
                print ("mbm.fc16 test write register success")
        end

        dataread_16bitTable,stat2 = mbm.fc3(L1,node,reg_addr,reg_cnt,timeout)
        if stat2==1 then
                print ("mbm.fc3 test read register success")
                i=0
                while (i<reg_cnt) do
                        i=i+1
                        print(dataread_16bitTable[i])
                end
        end

        mbm.disconnect(L1)
end
mbm.close(L1)
```

G. CANbus functions

Initialise on-board CANbus interface (specs: CAN 2.0A/B)

can_open(baud_rate)
baud_rate = your target CANbus system baud rate eg 50000,100000,125000,250000,500000 or 1000000
Maximum CAN baudrate supported by G3 is 1Mbps.
This must be executed once before any CANbus function call.

Sending CANbus messages

can_send(canid,extended,data_send)

canid	=000-7FF (3 hex chars for CAN 2.0A (standard))
	=00000000-1FFFFFFF (8 hex chars for CAN 2.0B (extended))
extended	=0 (11-bit message ID for CAN 2.0A (standard))
	=1 (29-bit message ID for CAN 2.0B (extended))
data_send	=ASCII hex chars of value 1-byte (eg 1A) to 8-bytes (eg 1A2B3C4D5E6F7A8B)
	Option to separate by '.' , e.g. 11.22.33.44.55.66.77.88

Receiving CANbus messages

data_receive = can_receive(canid,canmask,timeout)
returns 0 when no validated CANbus message received on timeout

canid	=000-7FF (3 hex chars for CAN 2.0A (standard))
	=00000000-1FFFFFFF (8 hex chars for CAN 2.0B (extended))
canmask	=000-7FF (bit value 0 mask out message)
	=00000000-1FFFFFFF (bit value 0 mask out message)
timeout	=receive timeout (seconds)

Mask usage example:
canid=0x5A0, canmask=0x7FF
canid & canmask = 0x5A0
Above mask will receive message with ID 0x5A0

canid=0x280, canmask=0x280
canid & canmask = 0x280
Above mask will receive message with ID 0x280 to 0x28F.

Storing CANbus messages into buffer

can_storebuff(canid,canmask)
This must be executed once to store CANbus messages into buffer.

canid	=000-7FF (3 hex chars for CAN 2.0A (standard))
	=00000000-1FFFFFFF (8 hex chars for CAN 2.0B (extended))
canmask	=000-7FF (bit value 0 mask out message)
	=00000000-1FFFFFFF (bit value 0 mask out message)

Reading CANbus messages from buffer

can_buff = can_readbuff()
returns buffer content and buffer emptied after reading

Close the CANbus interface

can_close()
Close the CANbus interface.

Example:

```
require “G3”
can_open(125000)            --baud rate=125000bps
sleep(0.2)

id1=”280”                   --canid=0x280
extended1=0                 --standard mode
data1=”AABBCC”              --data=0xAA,0xBB,0xCC
can_send(id1,extended1,data1)

id2="1F177678"              --canid=0x1F177678
extended2=1                 --extended mode
data2="11.22.33.44.55.66.77.88"   --data=0x11,0x22,0x33,0x44,0x55,0x66,0x77,0x88
can_send(id2,extended2,data2)

canid={}
canmask={}
canid[1]="111";canid[2]="222";canid[3]="333";canid[4]="444";canid[5]="555"
canmask[1]="7FF";canmask[2]="7FF";canmask[3]="7FF";canmask[4]="7FF";canmask[5]="7FF";
timeout=10          --seconds

can_storebuff(canid, canmask)       --store can messages to buffer

while (read_IN()==1) do
        can_buff=can_readbuff()     --read messages from buffer
        print(can_buff)
        sleep(1)
end

can_close()
```

H. Zigbee functions

zigbee_status()
returns 1 if zigbee port found
returns 0 if zigbee port not found

zigbee_open(speed)
opens the zigbee port
speed = 19200 (for Telegesis ETRX357 Zigbee adapter)

zigbee_send(message)
message = valid AT commands send to slave zigbee device

data_receive=zigbee_read(timeout_sec)
returns 0 when no Zigbee message received
timeout_sec =receive timeout (seconds)

zigbee_close()
close the zigbee port

Zigbee report attributes message comprise of ZCL header and attributes report.
Typical ZCL reporting :
RX:D55A,0104,02,0A,0702,0D:<18><F3><0A><0A><E1><21><A5><5C><43><E1><21><85><13>
Note that the attributes report with <xx> denotes the raw data value received. Eg <18>=0x18
Function zigbee_read() will convert the raw data value to ascii code. Eg <18> will be reported as "18"
RX:D55A,0104,02,0A,0702,0D:18F30A0AE121A55C43E1218513

Example for read attribute:

```
require "G3"

hexmesg=string.char(0x00,0x31,0x00,0x0A,0xE1,0x0D)      --attribute id=0xE10A
message="AT+UCASTB:05,D55A\r"..hexmesg                  --node id=D55A

if zigbee_status() then
        zigbee_open(19200)
        zigbee_send(message)
        repeat
                data_receive=zigbee_read(10)            --timeout=10sec
        until data_receive~=0
        print (data_receive)
        zigbee_close()
else
print ("zigbee port not found")
end
```

I. GPS function

This function is using the GPS capability built into the cellular module of Huawei model MU-609 (3G/HSPA+). The external GPS antenna (non-active type) is an optional item.

gps_start(interval)
interval = GPS positioning time interval (1-1800 seconds)
function initialize the GPS to standalone mode (non-assist) and start positioning.

gps_stop()
stops the GPS positioning routine in the wireless module

gps_read(timeout)
returns 0 when there is no gps position reported
returns a valid gps position report (NMEA type GPGGA and GPRMC)
timeout =receive timeout (seconds)

Example:

```
require "G3"
interval=10
gps_start(interval)
while (read_IN()==1) do                    --eg connect INPUT to generator switch
        sleep(1)
        gps=gps_read(10)                   --timeout=10sec
        print(gps)
end
gps_stop()
```

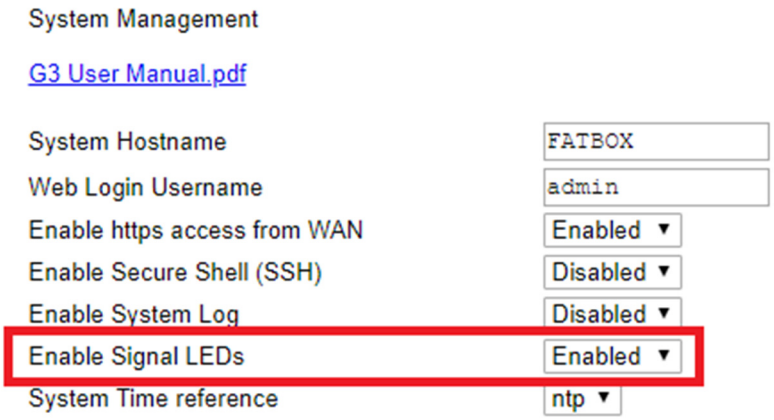
J. LUA script debug via SSH or CONSOLE (extended port version)

You can connect via SSH or CONSOLE to edit/run scripts and view debug information.
Just enable SSH, reboot and connect via SSH as 'root' and using same password as web login.
When connected, you can run your Lua scripts as below example.
/# lua /user/user.lua
Note that the Lua samples are located in the folder /user/samples/lua.
/# lua /user/samples/lua/chkdip.lua

K. SETUP REQUIREMENT for specific applications

Setup G3 router for SMS reboot application

For custom Lua application making use of SMS to reboot, it is recommended to disable the 'Signal LEDs' as shown below. Note that for build-in SMS reboot function at web config, 'signal LEDs' is automatically disabled.



Setup G3 router for SMS online/offline application

For custom Lua application making use of SMS to trigger ppp online/offline, it is recommended to

- Disable the 'Signal LEDs' as shown above.
- Disable the 'Reboot on Ping Failure' as shown below.
- Disable the 'Reboot on 3G Data Failure' as shown below.