深圳市拓普微科技开发有限公司

SHENZHEN TOPWAY TECHNOLOGY CO., LTD.

SMART LCD

Lua Functions Reference

制定:	复核:	审批:
日期: 2019-12-12	日期:	日期:

版本	描述	制定/修订	日期
0.1	- 初版	胡楚斌	2019-12-12

Table of Content

1 概述

Smart LCD 支持 Lua 脚本功能，本文说明了 hmt 库函数功能及使用方法。其他 Lua 编程语法及标准库函数功能请参考 Lua 官方手册。

为了满足部分用户的开发需求，Smart LCD 加入了 Lua 扩展式程序语言(又称 Lua 脚本语言)，Smart LCD 精减了标准 Lua5.3 的部分功能，并添加了 hmt 库函数接口以及回调处理函数（luamain、tpkhook、pagechangehook）便于用户灵活使用 Smart LCD。使用户能够自由的开发，定制并完善自己需要的功能。暂不支持 OS 操作及文件 IO 操作。

2 Smart LCD 的变量类型说明

简称	名称	内存限定	编码&地址范围
VP_STR	字符串变量 String Variable	1024(MAX)*(127+1)byte	0x000000~0x01FF80
VP_N16	16 位数字变量 16Bit Integer Variable	32512(MAX)*(2)byte	0x080000~0x08FDFE
VP_N32	32 位数字变量 32Bit Integer Variable	16128(MAX)*(4)byte	0x020000~0x02FEFC
VP_N64	64 位数字变量 64Bit Integer Variable	7936(MAX)*(8)byte	0x030000~0x03F7F8
VP_SYS	系统寄存器变量 System Register Variable	256(MAX)*(1)byte	0xFFFF00~0xFFFFFF

3 hmt 库函数说明

3.1 hmt 库函数列表

函数类型	函数名	功能描述
变量读写函数	hmt.readvpstr(addr)	读取字符串变量数据,返回字符串数据
	hmt.writevpstr(addr, string)	写入字符串变量数据
	hmt.readvp16(addr)	读取 16 位数字变量数据，返回整型数值
	hmt.writevp16(addr, value)	写入 16 位数字变量数据
	hmt.readvp32(addr)	读取 32 位数字变量数据，返回整型数值
	hmt.readvp32f(addr)	读取 32 位数字变量数据，返回浮点型数值
	hmt.writevp32(addr, value)	写入 32 位数字变量数据
	hmt.readvp64(addr)	读取 64 位数字变量数据，返回整型数值
	hmt.readvp64f(addr)	读取 64 位数字变量数据，返回浮点型数值
	hmt.writevp64(addr, value)	写入 64 位数字变量数据
	hmt.readvpreg(addr)	读取系统寄存器变量数据，返回整型数值
	hmt.writevpreg(addr, value)	写入系统寄存器变量数据
系统函数	hmt.changepage(pageid)	改变模块当前显示的页面
	hmt.readpage()	读取模块当前显示的页面，返回整型数值
	hmt.gettick()	获取系统运行时间，返回整型数值
	hmt.delayms(time)	系统延时，单位毫秒
	hmt.runcmd(table, num)	执行 TOPWAY 内部指令，不需要指令帧头帧尾
	hmt.bypass(is)	设置串口 0 是否禁用 TOPWAY 协议
	hmt.crc16calc(table, num)	计算数据 CRC 校验结果
串口 0 数据收发函数	hmt.uartisempty()	查询串口 0 接收区是否存在数据，返回值 1 表示 buf 为空，0 则反之
	hmt.uartclearbuf()	清除串口 0 接收 buf 中的数据
	hmt.getchar()	从串口 0 接收区获取一个字节数据，返回一个整型数据
	hmt.putchar(char)	使用串口 0 发送一个字节数据
	hmt.uartsendbytes(table, num)	使用串口 0 向外发送数据
	hmt.uartlock()	使用串口 0 前调用，获取锁。 与 hmt.uartunlock()成对使用
	hmt.uartunlock()	使用串口 0 后调用，释放锁。 与 hmt.uartlock()成对使用
串口 1 数据收发函数	hmt.uart1isempty()	查询串口 1 接收区是否存在数据，返回值 1 表示 buf 为空，0 则反之
	hmt.uart1clearbuf()	清除串口 1 接收 buf 中的数据
	hmt.uart1getchar()	从串口 1 接收区获取一个字节数据，返回一个整型数据
	hmt.uart1putchar(char)	使用串口 1 发送一个字节数据
	hmt.uart1sendbytes(table, num)	使用串口 1 向外发送数据
	hmt.uart1lock()	使用串口 1 前调用，获取锁。 与 hmt.uart1unlock()成对使用
	hmt.uart1unlock()	使用串口 1 后调用，释放锁。 与 hmt.uart1lock()成对使用

3.2 变量读写函数

3.2.1 hmt.readvpstr

函数原型	hmt.readvpstr(addr)
功能描述	读取模块中的字符串变量(VP_STR)数据
参数传入	addr 为字符串变量的地址 参考变量地址说明#2
返回值	字符串（string）
示例	local strvalue = ‘’ strvalue = hmt.readvpstr(0x00080) print(“strvalue=”, strvalue)
示例说明	预先写入字符串变量地址 0x000080 数据“TOPWAY” 执行结果： strvalue = TOPWAY

3.2.2 hmt.writevpstr

函数原型	hmt.writevpstr(addr, string)
功能描述	写入数据至模块中的字符串变量(VP_STR)
参数传入	addr 为字符串变量的地址，string 为需要写入的字符串数据 参考变量地址说明#2
返回值	无返回
示例	local strvalue = ‘ShenZhen’ hmt.writevpstr(0x00080, strvalue) print(“0x000080:”, hmt.readvpstr(0x000080))
示例说明	执行结果： 0x000080:ShenZhen

3.2.3 hmt.readvp16

函数原型	hmt.readvp16(addr)
功能描述	读取模块中的 16 位数字变量(VP_N16)数据
参数传入	addr 为 16 位数字变量的地址 参考变量地址说明#2
返回值	整型数值（integer）
示例	local N16value = 0 N16value = hmt.readvp16(0x080002) print(“N16value=”, N16value)
示例说明	预先写入 16 位数字变量地址 0x080002 数据“24” 执行结果： N16value = 24

3.2.4 hmt.writevp16

函数原型	hmt.writevp16(addr, value)
功能描述	写入数据至模块中的 16 位数字变量(VP_N16)
参数传入	addr 为 16 位数字变量的地址，value 为需要写入的数据 参考变量地址说明#2
返回值	无返回
示例	local N16value = 123 hmt.writevp16(0x080002, N16value) print("0x080002:", hmt.readvp16(0x080002))
示例说明	执行结果： 0x080002:123

3.2.5 hmt.readvp32

函数原型	hmt.readvp32(addr)
功能描述	读取模块中的 32 位数字变量(VP_N32)数据
参数传入	addr 为 32 位数字变量的地址 参考变量地址说明#2
返回值	整型数值（integer）
示例	local N32value = 0 N32value = hmt.readvp32(0x020004) print("N32value=", N32value)
示例说明	预先写入 32 位数字变量地址 0x020004 数据"0x42110000" 执行结果： N32value = 1108410368

3.2.6 hmt.readvp32f

函数原型	hmt.readvp32f(addr)
功能描述	读取模块中的 32 位数字变量(VP_N32)数据
参数传入	addr 为 32 位数字变量的地址 参考变量地址说明#2
返回值	浮点型数值（float）
示例	local N32value = 0.0 N32value = hmt.readvp32f(0x020004) print("N32value=", N32value)
示例说明	预先写入 32 位数字变量地址 0x020004 数据"0x42110000" 执行结果： N32value = 36.25

3.2.7 hmt.writevp32

函数原型	hmt.writevp32(addr, value)
功能描述	写入数据至模块中的 32 位数字变量(VP_N32)
参数传入	addr 为 32 位数字变量的地址，value 为需要写入的数据 参考变量地址说明#2
返回值	无返回
示例	local N32value = 54321 hmt.writevp32(0x020004, N32value) print("0x020004:", hmt.readvp32(0x020004))
示例说明	执行结果： 0x020004:54321

3.2.8 hmt.readvp64

函数原型	hmt.readvp64(addr)
功能描述	读取模块中的 64 位数字变量(VP_N64)数据
参数传入	addr 为 64 位数字变量的地址 参考变量地址说明#2
返回值	整型数值（integer）
示例	local N64value = 0 N64value = hmt.readvp64(0x030000) print("N64value=", N64value)
示例说明	预先写入 64 位数字变量地址 0x030000 数据"0x405D97AE147AE148" 执行结果： N64value = 4638029965185179976

3.2.9 hmt.readvp64f

函数原型	hmt.readvp64f(addr)
功能描述	读取模块中的 64 位数字变量(VP_N64)数据
参数传入	addr 为 64 位数字变量的地址 参考变量地址说明#2
返回值	浮点型数值（double）
示例	local N64value = 0.0 N64value = hmt.readvp64f(0x030000) print("N64value=", N64value)
示例说明	预先写入 64 位数字变量地址 0x030000 数据"0x405D97AE147AE148" 执行结果： N64value = 118.37

3.2.10 hmt.writevp64

函数原型	hmt.writevp64(addr, value)
功能描述	写入数据至模块中的 64 位数字变量(VP_N64)
参数传入	addr 为 64 位数字变量的地址，value 为需要写入的数据 参考变量地址说明#2
返回值	无返回
示例	local N64value = 123456789 hmt.writevp64(0x030000, N64value) print("0x030000:", hmt.readvp64(0x030000))
示例说明	执行结果： 0x030000: 123456789

3.2.11 hmt.readvpreg

函数原型	hmt.readvpreg(addr)
功能描述	读取模块中的系统寄存器变量(VP_SYS)数据
参数传入	addr 为系统寄存器变量的地址 参考变量地址说明#2
返回值	整型数值（integer）
示例	local Regvalue = 0 Regvalue = hmt.readvpreg(0xFFFF20) print("Regvalue=", Regvalue)
示例说明	预先写入系统寄存器变量地址 0xFFFF20 数据"8" 执行结果： Regvalue = 8

3.2.12 hmt.writevpreg

函数原型	hmt.writevpreg(addr, value)
功能描述	写入数据至模块中的系统寄存器变量(VP_SYS)
参数传入	addr 为系统寄存器变量的地址，value 为需要写入的数据 参考变量地址说明#2
返回值	无返回
示例	local Regvalue = 18 hmt.writevpreg(0xFFFF20, Regvalue) print("0xFFFF20:", hmt.readvpreg(0xFFFF20))
示例说明	执行结果： 0xFFFF20:18

3.3 系统函数

3.3.1 hmt.changepage

函数原型	hmt.changepage(pageid)
功能描述	切换模块当前显示页面
参数传入	pageid 为需要显示页面的 ID 号 ID 号范围 0~999，根据实际工程中存在的页面 ID 适当传参
返回值	无返回
示例	hmt.changepage(0x02)
示例说明	执行结果： 模块正常显示页面 2

3.3.2 hmt.readpage

函数原型	hmt.readpage()
功能描述	读取模块当前显示页面的 ID
参数传入	无参数传入
返回值	整型数值（integer）
示例	local pageid = 0 pageid = hmt.readpage() print("pageid=", pageid)
示例说明	执行结果： pageid=3 （工程当前显示页面的 ID 为 3）

3.3.3 hmt.gettick

函数原型	hmt.gettick()
功能描述	获取系统运行时间，单位 ms
参数传入	无参数传入
返回值	整型数值（integer）
示例	local worktime = 0 worktime = hmt.gettick() print("worktime=", worktime)
示例说明	执行结果： worktime=334520 （表示当前模块运行了 334520ms）

3.3.4 hmt.delayms

函数原型	hmt.delayms(time)
功能描述	系统延时函数
参数传入	time 为需要延时的时长，单位 ms
返回值	无返回值
示例	hmt.delayms(200)
示例说明	执行结果： 系统延时 200ms

3.3.5 hmt.runcmd

函数原型	hmt.runcmd(table, num)
功能描述	执行 TOPWAY 内部指令，不需要指令帧头帧尾
参数传入	table 为需要执行指令的字节序，num 为需要发送字节序的数量
返回值	无返回值
示例	local cmdtab = {0x5F, 0x04} hmt.runcmd(cmdtab, 2)
示例说明	请参考 TOPWAY 指令手册"0x5F"为背光控制命令码；"0x04"为背光亮度值； 背光范围"0x00~0x3F" 执行结果： 模块背光变暗

3.3.6 hmt.bypass

函数原型	hmt.bypass(is)
功能描述	设置串口 0 是否禁用 TOPWAY 协议
参数传入	is 为 0 时使用 TOPWAY 协议指令，is 为 1 时不使用 TOPWAY 协议指令； 默认状态为使用 TOPWAY 协议指令
返回值	无返回值
示例	hmt.bypass(1)
示例说明	使用串口发送 TOPWAY 指令（AA 5F 03 CC 33 C3 3C）试图改变模块背光亮度 执行结果： 模块未执行指令,背光没有变暗

3.3.7 hmt.crc16calc

函数原型	hmt.crc16calc(table, num)
功能描述	计算数据 CRC 校验结果，CRC(16:8005 reflected)校验参考#附录，校验结果低位在前
参数传入	table 为需要校验数据的集合，num 为需要校验的数据长度
返回值	整型数值（integer）
示例	local padcmd = {0x82, 0x00, 0x08, 0x05} local crcreturn = 0 crcreturn = hmt.crc16calc(padcmd, 4) print("crcreturn=", crcreturn)
示例说明	执行结果： crcreturn = 61343

3.4 串口 0 数据收发函数

串口 0 为模块默认通信串口，串口 1 为第二串口，Lua 打印函数 print 默认输出至串口 0。

3.4.1 hmt.uartisempty

函数原型	hmt.uartisempty()
功能描述	查询串口 0 接收区 buf 是否存在数据
参数传入	无参数传入
返回值	返回值 1 表示串口 0 接收区为空不存在数据，0 表示串口 0 接收区存在数据
示例	local Uart0state = 0 Uart0state = hmt.uartisempty() print("Uart0state=", Uart0state)
示例说明	执行结果： Uart0state = 1

3.4.2 hmt.uartclearbuf

函数原型	hmt.uartclearbuf()
功能描述	清除串口 0 接收区的数据
参数传入	无参数传入
返回值	无返回值
示例	hmt.uartclearbuf() print("Uart0state=", hmt.uartisempty())
示例说明	执行结果： Uart0state = 1

3.4.3 hmt.getchar

函数原型	hmt.getchar()
功能描述	从串口 0 接收区获取一个字节数据
参数传入	无参数传入
返回值	整型数值（integer）
示例	local getdata = 0 getdata = hmt.getchar() print("getdata=", getdata)
示例说明	使用串口助手发送 16 进制数据"0x99" 执行结果： getdata = 153

3.4.4 hmt.putchar

函数原型	hmt.putchar(char)
功能描述	使用串口 0 发送一个字节数据
参数传入	char：发送的数据 此处传值的数据应为数字类型，不支持字符串如：'C' "C"
返回值	无返回值
示例	hmt.putchar(0x41) hmt.putchar(66)
示例说明	执行结果： 串口助手接收 ASCII 字符："AB"；或接收 HEX 数据："41 42"

3.4.5 hmt.uartsendbytes

函数原型	hmt.uartsendbytes(table, num)
功能描述	使用串口 0 向外发送数据
参数传入	table 为需要发送的数据集合，num 为需要发送数据的数量
返回值	无返回值
示例	local sendbuff = {0xab, 0xcd, 0xef} hmt.uartsendbytes(sendbuff, 3)
示例说明	执行结果： 串口助手接收数据："ab cd ef"

3.4.6 hmt.uartlock

函数原型	hmt.uartlock()
功能描述	使用串口 0 前调用，获取锁，与 hmt.uartunlock()成对使用
参数传入	无参数传入
返回值	无返回值
示例	hmt.uartlock() hmt.putchar(0xaa) hmt.putchar(0x02) hmt.putchar(0x31) hmt.uartunlock()
示例说明	通过串口 0 连续发送："AA 02 31"

3.4.7 hmt.uartunlock

函数原型	hmt.uartunlock()
功能描述	使用串口 0 后调用，释放锁，与 hmt.uartlock()成对使用
参数传入	无参数传入
返回值	无返回值
示例	参考 hmt.uartlock()
示例说明	略

3.5 串口 1 数据收发函数

串口 0 为模块默认通信串口，串口 1 为第二串口，Lua 打印函数 print 默认输出至串口 0。

3.5.1 hmt.uart1isempty

函数原型	hmt.uart1isempty()
功能描述	查询串口 1 接收区 buf 是否存在数据
参数传入	无参数传入
返回值	返回值 1 表示串口 1 接收区为空不存在数据，0 表示串口 1 接收区存在数据
示例	local Uart1state = 0 Uart1state = hmt.uart1isempty() print("Uart1state=", Uart1state)
示例说明	执行结果： Uart1state = 1

3.5.2 hmt.uart1clearbuf

函数原型	hmt.uart1clearbuf()
功能描述	清除串口 1 接收区的数据
参数传入	无参数传入
返回值	无返回值
示例	hmt.uart1clearbuf() print("Uart1state=", hmt.uart1isempty())
示例说明	执行结果： Uart1state = 1

3.5.3 hmt.uart1getchar

函数原型	hmt.uart1getchar()
功能描述	从串口 1 接收区获取一个字节数据
参数传入	无参数传入
返回值	整型数值（integer）
示例	local getdata = 0 getdata = hmt.uart1getchar() print("getdata=", getdata)
示例说明	使用串口助手发送 16 进制数据 "aa" 执行结果： getdata = 170

3.5.4 hmt.uart1putchar

函数原型	hmt.uart1putchar(char)
功能描述	使用串口 1 发送一个字节数据
参数传入	char：发送的数据 此处传值的数据应为数字类型，不支持字符串如：'C' "C"
返回值	无返回值
示例	hmt.uart1putchar(0x61) hmt.uart1putchar(97)
示例说明	执行结果： 串口助手接收 ASCII 字符："ab"；或接收 HEX 数据："61 62"

3.5.5 hmt.uart1sendbytes

函数原型	hmt.uart1sendbytes(table, num)
功能描述	使用串口 1 向外发送数据
参数传入	table 为需要发送的数据集合，num 为需要发送数据的数量
返回值	无返回值
示例	local sendbuff = {0xab, 0xcd, 0xef} hmt.uart1sendbytes(sendbuff, 3)
示例说明	执行结果： 串口助手接收：“ab cd ef”

3.5.6 hmt.uart1lock

函数原型	hmt.uart1lock()
功能描述	使用串口 1 前调用，获取锁，与 hmt.uart1unlock()成对使用
参数传入	无参数传入
返回值	无返回值
示例	hmt.uart1lock() hmt.uart1putchar(0xaa) hmt.uart1putchar(0x02) hmt.uart1putchar(0x32) hmt.uart1unlock()
示例说明	通过串口 1 连续发送：“AA 02 32”

3.5.7 hmt.uart1unlock

函数原型	hmt.uart1unlock()
功能描述	使用串口 1 后调用，释放锁，与 hmt.uart1lock()成对使用
参数传入	无参数传入
返回值	无返回值
示例	参考 hmt.uart1lock()
示例说明	略

4 回调函数

4.1.1 luamain

函数原型	luamain = function(void)
功能描述	循环函数，周期执行，单位 10ms
参数传入	无参数传入
返回值	无返回值
示例	luamain = function(void) print(“TOPWAY”) return 0 end
示例说明	执行结果： 串口循环打印 “TOPWAY”

4.1.2 tpkhook

函数原型	tpkhook = function(page, id, state)
功能描述	触摸键回调函数 ，当有触摸键操作时，调用此函数，并传入触摸键 ID 信息
参数传入	触摸键的页面 ID，触摸键 ID，触摸键状态 页面 ID，触摸键 ID 可在开发工具中查看工程具体信息； 触摸键状态有按下、移出和释放三种状态，返回值分别为 1、2、0
返回值	1：不执行内部默认处理 0：执行内部默认处理
示例	tpkhook = function(page, id, state) print(“PageID=”, page, ”TPKID=”, id, ”state=”, state) return 0 end
示例说明	操作：按下页面 0 中 ID 为 2 的触摸键 执行结果： 串口接收信息“PageID=0 TPKID=2 state=1”

4.1.3 pagechangehook

函数原型	pagechangehook = function(pageid)
功能描述	页面跳转回调函数 ，当有跳页操作时调用此函数，并传入跳页后的页面 ID
参数传入	目标页面 ID，页面 ID 可在开发工具中查看工程具体信息
返回值	1：不执行页面跳转 0：执行页面跳转
示例	pagechangehook = function(pageid) print(“PageID=”, pageid) return 0 end
示例说明	操作：跳转至页面 1 执行结果： 串口接收信息 “PageID=1”

应用范例 – 模块闹钟功能实现

功能描述：

模块上电获取用户设置的闹钟时间（时分秒），用户设置闹钟时间使用 3 个 16 位数字变量控制。不进行闹钟时间设置则使用默认闹钟，默认闹钟时间为 07:20:00。闹铃通过内部指令控制蜂鸣器实现。

```
clockhour_0 = 0x7
clockminute_0 = 0x14
clocksecond_0 = 0x0
endclock = 0
luamain = function(void)

  if((hmt.readvpreg(0xffff13) == clockhour_0) and (hmt.readvpreg(0xffff14) ==
  clockminute_0) and (hmt.readvpreg(0xffff15) == clocksecond_0))then
    endclock = hmt.gettick() + 60000
  end

  if(endclock > hmt.gettick())then
    local buzzercmd = {0x7a, 0x01, 0x01, 0x01, 0x05, 0x08}
    hmt.runcmd(buzzercmd,6)
    hmt.delayms(200)
  end

return 0
end

tpkhook = function(page, id, state)

  if((page == 1)and(id == 2)and(state == 1))then
    clockhour_0 = hmt.readvp16(0x080002)
    clockminute_0 = hmt.readvp16(0x080004)
    clocksecond_0 = hmt.readvp16(0x080006)
  end
  if((page == 1)and(id == 4)and(state == 1))then
    endclock = hmt.gettick()
  end

return 0
end

pagechangehook = function(pageid)

return 0
end
```

*注意：脚本程序中不可使用等待循环

附录：CRC Calculate

```
uint16_t const CRC16[256]={
/* 16: 8005 reflected */
  0x0000,0xc0c1,0xc181,0x0140,0xc301,0x03c0,0x0280,0xc241,
  0xc601,0x06c0,0x0780,0xc741,0x0500,0xc5c1,0xc481,0x0440,
  0xcc01,0x0cc0,0x0d80,0xcd41,0x0f00,0xcfc1,0xce81,0x0e40,
  0x0a00,0xcac1,0xcb81,0x0b40,0xc901,0x09c0,0x0880,0xc841,
  0xd801,0x18c0,0x1980,0xd941,0x1b00,0xdbc1,0xda81,0x1a40,
  0x1e00,0xdec1,0xdf81,0x1f40,0xdd01,0x1dc0,0x1c80,0xdc41,
  0x1400,0xd4c1,0xd581,0x1540,0xd701,0x17c0,0x1680,0xd641,
  0xd201,0x12c0,0x1380,0xd341,0x1100,0xd1c1,0xd081,0x1040,
  0xf001,0x30c0,0x3180,0xf141,0x3300,0xf3c1,0xf281,0x3240,
  0x3600,0xf6c1,0xf781,0x3740,0xf501,0x35c0,0x3480,0xf441,
  0x3c00,0xfcc1,0xfd81,0x3d40,0xff01,0x3fc0,0x3e80,0xfe41,
  0xfa01,0x3ac0,0x3b80,0xfb41,0x3900,0xf9c1,0xf881,0x3840,
  0x2800,0xe8c1,0xe981,0x2940,0xeb01,0x2bc0,0x2a80,0xea41,
  0xee01,0x2ec0,0x2f80,0xef41,0x2d00,0xedc1,0xec81,0x2c40,
  0xe401,0x24c0,0x2580,0xe541,0x2700,0xe7c1,0xe681,0x2640,
  0x2200,0xe2c1,0xe381,0x2340,0xe101,0x21c0,0x2080,0xe041,
  0xa001,0x60c0,0x6180,0xa141,0x6300,0xa3c1,0xa281,0x6240,
  0x6600,0xa6c1,0xa781,0x6740,0xa501,0x65c0,0x6480,0xa441,
  0x6c00,0xacc1,0xad81,0x6d40,0xaf01,0x6fc0,0x6e80,0xae41,
  0xaa01,0x6ac0,0x6b80,0xab41,0x6900,0xa9c1,0xa881,0x6840,
  0x7800,0xb8c1,0xb981,0x7940,0xbb01,0x7bc0,0x7a80,0xba41,
  0xbe01,0x7ec0,0x7f80,0xbf41,0x7d00,0xbdc1,0xbc81,0x7c40,
  0xb401,0x74c0,0x7580,0xb541,0x7700,0xb7c1,0xb681,0x7640,
  0x7200,0xb2c1,0xb381,0x7340,0xb101,0x71c0,0x7080,0xb041,
  0x5000,0x90c1,0x9181,0x5140,0x9301,0x53c0,0x5280,0x9241,
  0x9601,0x56c0,0x5780,0x9741,0x5500,0x95c1,0x9481,0x5440,
  0x9c01,0x5cc0,0x5d80,0x9d41,0x5f00,0x9fc1,0x9e81,0x5e40,
  0x5a00,0x9ac1,0x9b81,0x5b40,0x9901,0x59c0,0x5880,0x9841,
  0x8801,0x48c0,0x4980,0x8941,0x4b00,0x8bc1,0x8a81,0x4a40,
  0x4e00,0x8ec1,0x8f81,0x4f40,0x8d01,0x4dc0,0x4c80,0x8c41,
  0x4400,0x84c1,0x8581,0x4540,0x8701,0x47c0,0x4680,0x8641,
  0x8201,0x42c0,0x4380,0x8341,0x4100,0x81c1,0x8081,0x4040,
};

static __inline uint16_t rshiftu16(uint16_t value, int nb)
{
  return (uint16_t)((value >> nb) & ~((( uint16_t) 0x8000) >> (nb-1)));
}
uint16_t crc16_calc(unsigned char *q, int len)
{
  uint16_t crc = 0xffff;
  while (len-- > 0)
    crc=(rshiftu16(crc,8) ^ CRC16[(crc ^ *q++) & 0xff]);
  return crc;
}
```