华成控制系统Lua帮助文档

概述

内置函数

C++ 注册的函数

```
--[[
    version:1.0.0.0
    更新日期：2021.11.20
    系统内置函数:可直接使用

    功能：写输出IO状态
    参数： board: 输出板,id：输出点  state:输出状态
    返回值： 无
    void hc_set_io_output(board,id,state)

    功能：读输出IO状态
    参数：board: 输出板,id:输出点
    返回值：输出点状态
    int hc_get_io_output(board,id)

    功能：读IO状态
    参数：board: 输入板，id:输入点
    返回值：输入点状态
    int hc_get_io_input(board,id)

    功能：写输出M值状态
    参数： board: 输出板,id：输出点  state:输出状态
    返回值： 无
    void hc_set_m_output(board,id,state)

    功能：读M值状态
    参数：board: 输出板,id:输出点
    返回值：输入点状态
    int hc_get_m_input(board,id)

    功能：设置系统报警
    参数：error_num 报警号 范围(9000 - 9999)
    返回值：无
    void hc_set_system_error(error_num)

    功能：获取系统报警
    参数：无
    返回值：系统报警号
    int hc_get_system_error()

    功能：获取系统时间
    参数：无
    返回值：系统绝对时间 单位ms
    uint64 hc_get_system_timer()

    功能：获取当前系统模式
    参数:无
    返回值：1：手动模式 2：自动模式 3：停止模式
    uint32 hc_get_system_mode()

    功能：写modbus保持寄存器
    参数：addr：保持寄存器地址  value：值
```

返回值：无
void write_modbus_hold_reg(addr,value)

功能：读modbus保持寄存器
参数：addr：保持寄存器地址
返回值：保持寄存器地址中的值
uint16 read_modbus_hold_reg(addr)

功能：写系统临时寄存器
参数：addr：系统临时寄存器（800~898,3000~4000） value：值
返回值：无
void hc_set_system_temp_reg(addr,value)

功能：读系统临时寄存器
参数：addr：系统临时寄存器（800~898,3000~4000）
返回值：临时寄存器中的值
uint32 hc_get_system_temp_reg(addr)

功能：获取轴的关节坐标
参数：轴id
返回值：对应轴的关节坐标 单位°/mm
float hc_get_axis_current_joint(axis_id)

功能：获取轴的世界坐标
参数：轴id
返回值：对应轴的世界坐标 单位°/mm
float hc_get_axis_current_pos(axis_id)

功能：通过tcp发送debug消息
参数：格式控制，变量若干
返回值：无
void hc_debug_printf(fmt, ...)

功能：获取本地ip地址
参数：无
返回值：ip地址的字符串 格式：xxx.xxx.xxx.xxx
void hc_get_local_ip_addr()

功能：配置tcp客户端
参数1：目标地址 格式：xxx.xxx.xxx.xxx
参数2：目标端口
参数3：重连时间 单位ms
返回值：无
void hc_tcp_config_client(str,int,int)

功能：配置tcp服务器
参数1：本地端口
返回值：无
void hc_tcp_config_server(int)

功能：通过tcp发送字符流消息，默认端口9781
参数：字符串
返回值：发送数据长度
int hc_debug_send_msg(str)

功能：通过tcp接收字符流消息，默认端口9781
参数：无
返回值：长度， 接收的信息
int,string hc_tcp_read_msg()

功能：通过tcp发送字节流消息，默认端口9782
参数：字节流array

返回值：发送数据长度
int hc_tcp_send_byte(array)

功能：通过tcp接收字节流消息，默认端口9782
参数：无
返回值：长度， 接收的字节流信息
int,array hc_tcp_read_byte()

功能：通过tcp发送字符流消息，默认端口9782
参数：字符串
返回值：发送数据长度
int hc_tcp_send_msg(str)

功能：通过tcp接收字符流消息，默认端口9782
参数：无
返回值：长度，接收的字符流信息
int,string hc_tcp_read_msg()

功能：获取计数器当前值
参数：计数器id
返回值： 计数器当前值
int hc_get_counter_current(int)

功能：获取计数器目标值
参数：计数器id
返回值： 计数器当前值
int hc_get_counter_target(int)

功能：设置计数器当前值
参数1：计数器id
参数2：设置的值
返回值： 0：设置失败 1：设置成功
int hc_set_counter_current(int，int)

功能：获取计时器当前值
参数：计时器id
返回值： 计时器当前值
int hc_get_timer_current(int)

功能：获取计时器目标值
参数：计时器id
返回值： 计时器当前值
int hc_get_timer_target(int)

功能：设置计时器当前值
参数1：计时器id
参数2：设置的值
返回值： 0：设置失败 1：设置成功
int hc_set_timer_current(int，int)

功能：获取输出IO当前沿信号
参数：id：输出点(0-159为Y输出 160-222为M值) type:沿信号类型 0：上升沿 1：下降沿
返回值： 当前IO上升沿/下降沿信号
int hc_get_io_output_edge(id,type)

功能：获取输入IO当前沿信号
参数：id：输出点(id:0-159为X输入IO) type:沿信号类型 0：上升沿 1：下降沿
返回值： 当前IO上升沿/下降沿信号
int hc_get_io_input_edge(id,type)

功能：计数器当前值+1
参数：id：计数器id

返回值： 无
void hc_set_counter_add(id)

功能：计数器当前值-1
参数：id：计数器id
返回值： 无
void hc_set_counter_dec(id)

功能：设置计时器目标值
参数1：计时器id
参数2：设置的值
返回值： 0：设置失败 1：设置成功
int hc_set_timer_target(id，target_cnt)

功能：设置计数器目标值
参数1：计数器id
参数2：设置的值
返回值： 0：设置失败 1：设置成功
int hc_set_counter_target(id，target_cnt)

功能：写系统临时寄存器
参数：addr：系统临时寄存器（4000~5999） value：值
返回值：无
void hc_set_system_temp_reg_16(addr,value)

功能：读系统临时寄存器
参数：addr：系统临时寄存器（4000~5999）
返回值：临时寄存器中的值
uint32 hc_get_system_temp_reg_16(addr)

功能：置位IO输入触发函数
参数：id：置位Io (id:0-159为X输入IO)
返回值： 无
int hc_set_input_trigger(id)

功能：复位IO输入触发函数
参数：id：复位Io (id:0-159为X输入IO)
返回值： 无
int hc_reset_input_trigger(id)

功能：置位IO输出触发函数
参数：id：置位Io (id:0-159为Y输出 160-222为M值)
返回值： 无
int hc_set_output_trigger(id)

功能：复位IO输出触发函数
参数：id：复位Io (id:0-159为Y输出 160-222为M值)
返回值： 无
int hc_reset_output_trigger(id)

功能：开始计时器
参数：id：计时器id
返回值： 无
int hc_start_timer(id)

功能：结束计时器
参数：id：计时器id
返回值： 无
int hc_stop_timer(id)

功能：读modbus保持寄存器，转换成64位数据
参数：addr1：保持寄存器地址1，addr2：保持寄存器地址2，addr3：保持寄存器地址3，addr4：保持寄存器地址4

返回值：根据读取到的addr1、2、3、4地址的值由低到高位转换成64位数据
uint64 hc_get_modbus_holdreg_64(addr1,addr2,addr3,addr4)

功能：注册Task
参数：task_name 任务名称(任务函数名) ； cycle_ms 任务扫描周期ms ;cycle_count 任务执行次数（0为不限次数）
返回值：0：注册成功，-1：注册失败
int hc_register_task(task_name,cycle_ms,cycle_count)

功能：延时ms 只能在task中时间，阻塞函数
参数：ms
返回值：无
void hc_task_delay(ms)

功能：获取IO点上升沿
参数：io_id：io点；check_time：检测时间ms ； filter_time：滤波时间ms(可选参数)
返回值：是否捕获到上升沿
bool hc_get_io_r_trig(io_id,check_time,filter_time)

功能：获取IO点下降沿
参数：io_id：io点；check_time：检测时间ms ； filter_time：滤波时间ms(可选参数)
返回值：是否捕获到下降沿
bool hc_get_io_r_trig(io_id,check_time,filter_time)

--]]

lua 函数

- 以下内容需要添加到main.lua文件中

```
-------------------------------------
-- 系统函数
-------------------------------------

-- 打印信息到 IDE 窗口
function hc_debug_printf(fmt, ...)
   local str = string.format(fmt, ...)
   local fun_name = debug.getinfo(1).name
   local line = debug.getinfo(1).currentline
   local info = "[ function: "..fun_name.."  line: "..line.." ]> "..str.."\n"
   hc_debug_send_msg(info)
end

function hc_delay_us(time)
   local socket = require("socket")
   socket.sleep(time / 1000000) -- 转换为秒
end

-- IO名字转id
-- 例:  X010->0  X047->31
function hc_io_name_to_id(name)
   local io_name = ""
   io_name = name
   local type = string.sub(io_name,1,1)
   if( type == 'x' or type == 'X' or type == 'Y' or type == 'y' or type == 'M' or type == 'm') then
       local io_id = ""
       local board_id = 0
       if(#io_name == 3) then
           io_id = string.sub(io_name,2,3)

       elseif (#io_name == 4) then
           io_id = string.sub(io_name,3,4)
           local borad = string.sub(io_name,2,2);
           board_id = tonumber(borad)

       end
       local id = tonumber(io_id,8) - 8 + board_id*32
       return id
   end
   return 0
end

-- 返回系统当前毫秒数
function GetSystemTime()
   return os.clock()* 1000
end

-- 通电延时定时器
TON = {
__ClassType = "class type",
__input_l = false,
__Q =false,
__init_s = false,
__start=false,
__start_time = 0,
__input_fun = {},
__time = 0,
}

function TON:New(o)
   o = o or {}
```

```
    o.__ClassType = "class type"
    o.mt = { __index = o}
    setmetatable(o, {__index = self})
    return o
end

function TON:Run(fun,time)
    if(self.__init_s == true) then
        local s = fun()
        if(s == true and self.__input_l == false) then
            self.__start = true;
            self.__start_time = GetSystemTime()
        elseif (s == false and self.__input_l == true) then
            self.__init_s = false
        end
        self.__input_l = s
    end

    if(self.__init_s == false) then
        self.__start = false
        self.__init_l = fun
        self.__time = time
        self.__Q = false
    end

    if(self.__start) then
        if(GetSystemTime() - self.__start_time >= self.__time ) then
            self.__Q = true
        end
    end

    self.__init_s = true;
    return self.__Q
end

-- 断电延时定时器
TOF = {
__ClassType = "class type",
__input_l = false,
__Q =false,
__init_s = false,
__start=false,
__start_time = 0,
__input_fun = {},
__time = 0,
}

function TOF:New(o)
    o = o or {}
    o.__ClassType = "class type"
    o.mt = { __index = o}
    setmetatable(o, {__index = self})
    return o
end

function TOF:Run(fun,time)
    if(self.__init_s == true) then
        local s = fun()
        if(s == false and self.__input_l == true) then
            self.__start = true;
            self.__start_time = GetSystemTime()
        elseif (s == true and self.__input_l == false) then
```

```
            self.__init_s = false
        end
        self.__input_l = s
    end

    if(self.__init_s == false) then
        self.__start = false
        self.__init_l = fun
        self.__time = time
        self.__Q = false
    end

    if(self.__start) then
        if(GetSystemTime() - self.__start_time >= self.__time ) then
            self.__Q = true
        end
    end

    self.__init_s = true;
    return self.__Q
end

-- 上升沿检测触发器
R_TRIG = {
    __ClassType = "class type",
    __input_l = false,
    __Q =false,
    __init_s = false,
    __input_fun = {},
    }

function R_TRIG:New(o)
    o = o or {}
    o.__ClassType = "class type"
    o.mt = { __index = o}
    setmetatable(o, {__index = self})
    return o
end

function R_TRIG:Run(fun)
    if(self.__init_s == true) then
        local s = fun()
        if(s == true and self.__input_l == false) then
            self.__Q = true
        elseif (s == false and self.__input_l == true) then
            self.__init_s = false
        end
        self.__input_l = s
    end

    if(self.__init_s == false) then
        self.__start = false
        self.__init_l = fun
        self.__Q = false
    end

    self.__init_s = true;
    return self.__Q
end

-- 下降沿检测触发器
F_TRIG = {
```

```
    __ClassType = "class type",
    __input_l = false,
    __Q =false,
    __init_s = false,
    __input_fun = {},
    }

function F_TRIG:New(o)
    o = o or {}
    o.__ClassType = "class type"
    o.mt = { __index = o}
    setmetatable(o, {__index = self})
    return o
end

function F_TRIG:Run(fun)
    if(self.__init_s == true) then
        local s = fun()
        if(s == true and self.__input_l == false) then
            self.__Q = true
        elseif (s == false and self.__input_l == true) then
            self.__init_s = false
        end
        self.__input_l = s
    end

    if(self.__init_s == false) then
        self.__start = false
        self.__init_l = fun
        self.__Q = false
    end

    self.__init_s = true;
    return self.__Q
end
```

例程