```
/*-----------------------------------------------------------------------------------------------------------------------
 *
 * Initialise the DDModbusMaster Finite State Machine
 * Serial port needs to be defined, an arduino UNO only has one hardware serial port, 'Serial0'.
 * Other Arduino boards such as Mega have four serial ports, 'Serial0' through 'Serial3'.
 * Or the Leonardo, Yún or pro micro (32u4 processor) where the the usb port is used
 * for pc <-> mcu communication and the other on board uart for RS485.
 * fnCallback is the function pointer for the onCallDone delegate.
 *
 * Charge Controller Selection List:
 *      0 - Simulator
 *
 *  Midnight Solar
 *      10  - CLASSIC all models 150, 200 and 250
 *
 *  Morningstar
 *      20  - TRISTAR_MPPT_30, TRISTAR_MPPT_45 and TRISTAR_MPPT_60
 */
#ifndef modbus_maps_H_
#define modbus_maps_H_                                          // header guard:
#include "DDModbusMaster.h"                                     // header for Modbus Master library:
#include "defines.h"                                            // header for definitions:

#define SLAVE_ID            1                                   // ID of slave device:
#define MB_BAUD             19200                               // baud rate, Classic = 19200, TriStar = 9600:
#define MB_POLL_RATE        250L                                // periodic scan rate:
#define MB_TIMEOUT          1000L                               // timeout to catch error conditions:
#define MB_RETRY            10                                  // total retry preset:
#define MB_TX_ENABLE_PIN    0xff                                // EIA-485 transmit pin enable, set to 0xff for RS-232 mode, no pin used:
#define LOW_CLAMP_VOLTS_SP  4                                   // scale * 10: limit number of 0.1 volts below charge controller setpoint:

#define CHARGE_CONTROLLER_TYPE 0                                // <<<<<< Make certain the correct charge controller is selected here >>>>>>

//-----------------------------------------------------------------------------------------------------------------------
//  const   type            variable            =   value;          comment
            uint16_t        readRegs[21];                       // modbus read registers array:

    struct reed_param_t                                         // structure to hold reed controller values:
    {
            int16_t         volt_setpoint;                      // * 10 calculated setpoint for PID to be driven to:
            int16_t         pid_sp_offset;                      // * 10 PID setpoint offset trim adjust, ie value of 2 equates to -0.2 volts below controller setpoint:
            int16_t         pid_sp_low_clamp;                   // * 10 PID setpoint offset lower clamp value, ie value of 4 equates to -0.4 volts below setpoint:
        }   reed_param;                                         // instance of structure:

    struct  mb_status_t                                         // structure to hold Modbus success or error values:
    {
            uint16_t        poll_count;
            uint16_t        poll_count_10K;
            uint16_t        timeouts;
            uint16_t        address;
            uint8_t         code;
        }   mb_status;                                          // instance of structure:

    struct  cc_param_t                                          // structure to hold usuable data from charge contriolller:
    {
            int16_t         vBattery_pv;                        // * 10 battery volts:
            int16_t         vArray_pv;                          // * 10 array volts:
            int16_t         vBattery_sp;                        // * 10 battery setpoint target:
            uint16_t        battery_amps;                       // * 10 charge controller charge current:
            uint16_t        kwh;                                // * 10 accumulated kWHrs through chareg controller:
            uint16_t        watts;                              // * 1  charge controller charge wattage:
            uint16_t        mode;                               // charge controller mode:
        }   cc_param;                                           // instance of struct:

            DDModbusMaster  charge_controller;                  // instance of DDModbusMaster class:

void onCallDone(Packet*, errorTypes, uint8_t);                  // forward declare the callback function here, not in main.h:

/*-----------------------------------------------------------------------------------------------------------------------
 * Charge Controller Templates:
 * Arduino Simulator for  Midnight Solar.
 */
#if (CHARGE_CONTROLLER_TYPE == 0)                               // charge controller selection number:
    #define REGISTER_OFFSET     4000                            // registers usually are in the range of 40001 - 49999 but need to be addressed starting from 0.
    #define REG_BATTERY_VOLTS   4000 - REGISTER_OFFSET          // register address in specific Modbus Function Code:
//  #define REG_BATTERY_SP      4020 - REGISTER_OFFSET          // battery setpoint voltage for any given charge mode (target volts):

    enum enum_mb_reg_packet_1                                   // Modbus registers for packet 1:
    {
        MB_vBattery_pv = REG_BATTERY_VOLTS,                     // battery voltage averaged over 1 second:
        MB_vArray_pv,                                           // pv array voltage averaged over 1 second:
        MB_iBattery_pv,                                         // battery current averaged over 1 second:
        MB_kwhr_pv,                                             // energy delivered to battery per day:
        MB_watts_pv,                                            // average power to battery:
        MB_mode_pv,                                             // charge controller state:
        MB_vBattery_sp,                                         // charge controller battery setpoint target:
        TOTAL_PACKET_1                                          // calculate total registers for packet:
    };

    enum enum_mb_reg_packet_2                                   // Modbus registers for packet 2:
    {
        MB_cc_pwm,                                              // pwm value to send back for simulation:
        TOTAL_PACKET_2                                          // calculate total registers for packet:
    };
    uint16_t    cc_pwm;

    enum enum_cc_state                                          // enumeration for specific charge controller state:
    {
        mode_resting,
        mode_absorb = 3,
        mode_bulkmppt,
        mode_float,
        mode_floatmppt,
        mode_equalise,
        mode_hypervoc = 10,
        mode_equlisemppt = 18
    }   controller_state;                                       // enumeration od charge controller states:

    enum enum_local_reg                                         // local register mappings:
    {
        vBattery_pv,
        vArray_pv,
        iBattery_pv,
        kwhr_pv,
        watts_pv,
        mode_pv,
        vBattery_sp,
        TOTAL_LOCAL_REG                                         // total number of local registers, alway at end:
    }   local_reg;

    unsigned int    regs[TOTAL_LOCAL_REG];                      // create array for local registers:

    enum enum_mb_packet                                         // Modbus packets:
    {
        PACKET1,                                                // packet to read charge controller data:
//      PACKET2,                                                // packet to send PWM for simulator:
        TOTAL_PACKETS                                           // total packet count:
    }mb_packet;

    Packet packets[TOTAL_PACKETS];                              // create array for configured packets
//  packetPointer   packet1 = &packets[PACKET1];                // pointer to packet, not required as access to the array can be made explicitly, packets[PACKET1].id = 2;

/*-------------------------------------------------------------------------------
 *  void mbPacketInit()
 *  DDModbusMaster packet and Modbus communications port initialisation:
 */
void mbPacketInit() {
//  Callback fnCallback = onCallDone;                           // function pointer for the DDModbusMaster callback method:
//  read seven registers starting from MB_vBattery_volts_pv and save results to the regs[] array starting at index vBattery_volts_pv
    charge_controller.modbus_construct(&packets[PACKET1], SLAVE_ID, READ_HOLDING_REGISTERS,
            MB_vBattery_pv,
            TOTAL_PACKET_1,
            vBattery_pv);

//  charge_controller.modbus_construct(&packets[PACKET2], SLAVE_ID, PRESET_SINGLE_REGISTER,
//          MB_cc_pwm,
//          TOTAL_PACKET_2,
//          cc_pwm);
////            , B00000100);   //for one shot

    charge_controller.modbus_configure(MB_SERIAL_PORT, MB_BAUD, SERIAL_8N1, MB_TIMEOUT, MB_POLL_RATE, MB_RETRY, MB_TX_ENABLE_PIN, packets, TOTAL_PACKETS, regs, onCallDone);
}
#endif /* CHARGE_CONTROLLER_NUMBER == 0 */

/*-----------------------------------------------------------------------------------------------------------------------
 * Charge Controller Templates:
 * Midnight Solar Classic series.
 */
#if (CHARGE_CONTROLLER_TYPE == 10)                              // charge controller selection number:
    #define REGISTER_OFFSET     4000                            // registers usually are in the range of 40001 - 49999 but need to be addressed starting from 0.
    #define REG_BATTERY_VOLTS   4115 - REGISTER_OFFSET          // register address in specific Modbus Function Code:
    #define REG_TEMP_BATTERY    4132 - REGISTER_OFFSET
    #define REG_BATTERY_SP      4244 - REGISTER_OFFSET          // battery setpoint voltage for any given charge mode (target volts):

    enum enum_mb_reg_packet_1                                   // Modbus registers for packet 1:
    {
        MB_vBattery_pv = REG_BATTERY_VOLTS,                     // battery voltage averaged over 1 second:
        MB_vArray_pv,                                           // pv array voltage averaged over 1 second:
        MB_iBattery_pv,                                         // battery current averaged over 1 second:
        MB_kwhr_pv,                                             // energy delivered to battery per day:
        MB_watts_pv,                                            // average power to battery:
        MB_mode_pv,                                             // charge controller state:
        TOTAL_PACKET_1 = MB_mode_pv - MB_vBattery_pv + 1        // calculate total registers for packet:
    };

    enum enum_mb_reg_packet_2                                   // Modbus registers for packet 2:
    {
        MB_vBattery_sp = REG_BATTERY_SP,                        // charge controller battery setpoint target:
        TOTAL_PACKET_2                                          // calculate total registers for packet:
    };

    enum enum_cc_state                                          // enumertion for specific charge controller state:
    {
        mode_resting,
        mode_absorb = 3,
        mode_bulkmppt,
        mode_float,
        mode_floatmppt,
        mode_equalise,
        mode_hypervoc = 10,
        mode_equlisemppt = 18
    }   controller_state;                                       // enumeration od charge controller states:

    enum enum_local_reg                                         // local register mappings:
    {
        vBattery_pv,
        vArray_pv,
        iBattery_pv,
        kwhr_pv,
        watts_pv,
        mode_pv,
        vBattery_sp,
        TOTAL_LOCAL_REG                                         // total number of local registers, alway at end:
    }   local_reg;

    unsigned int    regs[TOTAL_LOCAL_REG];                      // create array for local registers:

    enum enum_mb_packet                                         // Modbus packets:
    {
        PACKET1,
        PACKET2,
        TOTAL_PACKETS                                           // total packets:
    }mb_packet;
    Packet packets[TOTAL_PACKETS];                              // create array for configured packets

/*-------------------------------------------------------------------------------
 * DDModbusMaster packet and Modbus communications port initialisation:
 */
void mbPacketInit() {
    charge_controller.modbus_construct(&packets[PACKET1], SLAVE_ID, READ_HOLDING_REGISTERS,
            MB_vBattery_pv,
            TOTAL_PACKET_1,
            vBattery_pv);

    charge_controller.modbus_construct(&packets[PACKET2], SLAVE_ID, READ_HOLDING_REGISTERS,
            MB_vBattery_sp,
            TOTAL_PACKET_2,
            vBattery_sp);

    charge_controller.modbus_configure(MB_SERIAL_PORT, MB_BAUD, SERIAL_8N1, MB_TIMEOUT, MB_POLL_RATE, MB_RETRY, MB_TX_ENABLE_PIN, packets, TOTAL_PACKETS, regs, onCallDone);
}
#endif /* CHARGE_CONTROLLER_NUMBER == 10 */

/*-----------------------------------------------------------------------------------------------------------------------
 * Charge Controller Templates:
 * Morningstar TriStar MPPT Series.
 */
#if (CHARGE_CONTROLLER_TYPE == 20)                              // charge controller selection number:
    #define REGISTER_OFFSET     4000                            // registers usually are in the range of 40001 - 49999 but need to be addressed starting from 0.
    #define REG_V_PU_HI         4000 - REGISTER_OFFSET          //
    #define REG_V_PU_LO         4001 - REGISTER_OFFSET          //
    #define REG_I_PU_HI         4002 - REGISTER_OFFSET          //
    #define REG_I_PU_LO         4003 - REGISTER_OFFSET          //

    #define REG_BATTERY_VOLTS   4025 - REGISTER_OFFSET          //
    #define REG_ARRAY_VOLT      4028 - REGISTER_OFFSET          //

    #define REG_STATE           4051 - REGISTER_OFFSET          //
    #define REG_BATTERY_SP      4052 - REGISTER_OFFSET          //
    #define REG_KWHR            4057 - REGISTER_OFFSET          //
    #define REG_WATTS           4059 - REGISTER_OFFSET          //

    enum enum_mb_reg_packet_1                                   // Modbus registers for packet 1:
    {
        MB_scale_volts_high = REG_V_PU_HI,
        MB_scale_volts_low = REG_V_PU_LO,
        MB_scale_amps_high = REG_I_PU_HI,
        MB_scale_amps_low = REG_I_PU_LO,
        TOTAL_PACKET_1 = MB_scale_amps_low - MB_scale_volts_high + 1        // calculate total registers for packet:
    };

    enum enum_mb_reg_packet_2                                   // Modbus registers for packet 2:
    {
        MB_vBattery_pv = REG_BATTERY_VOLTS,                     // battery voltage averaged over 1 second:
        MB_vArray_pv = REG_ARRAY_VOLT,                          // pv array voltage averaged over 1 second:
        MB_iBattery_pv,                                         // battery current averaged over 1 second:
        TOTAL_PACKET_2 = MB_iBattery_pv - MB_vBattery_pv + 1    // calculate total registers for packet:
    };

    enum enum_mb_reg_packet_3                                   // Modbus registers for packet 3:
    {
        MB_mode_pv = REG_STATE,                                 // charge controller state:
        MB_vBattery_sp = REG_BATTERY_SP,                        // charge controller battery setpoint target:
        MB_kwhr_pv = REG_KWHR,                                  // energy delivered to battery per day:
        MB_watts_pv = REG_WATTS,                                // average power to battery:
        TOTAL_PACKET_3 = MB_watts_pv - MB_mode_pv + 1           // calculate total registers for packet:
    };

```

```
    enum enum_cc_state                                                  // enumertion for specific charge controller state:
    {
        mode_start,
        mode_night_check,
        mode_disconnect,
        mode_night,
        mode_fault,
        mode_mppt,
        mode_absorb,
        mode_float,
        mode_equalise,
        mode_slave,
        mode_resting,                                                   // false
        mode_bulkmppt,                                                  // false
        mode_floatmppt,                                                 // false
        mode_hypervoc,                                                  // false
        mode_equlisemppt,                                               // false
    }   controller_state;                                               // enumeration od charge controller states:

    enum enum_local_reg                                                 // local register mappings:
    {
        v_scaling_high,
        v_scaling_low,
        vBattery_pv,
        vArray_pv,
        iBattery_pv,
        kwhr_pv,
        watts_pv,
        mode_pv,
        vBattery_sp,
        TOTAL_LOCAL_REG                                                 // total number of local registers, alway at end:
    }   local_reg;

    unsigned int    regs[TOTAL_LOCAL_REG];                              // create array for local registers:
    float           scale_volts = 0.0;
    float           scale_amps = 0.0;

    enum enum_mb_packet                                                 // Modbus packets:
    {
        PACKET1,
        PACKET2,
        PACKET3,
        TOTAL_PACKETS                                                   // total packets:
    }mb_packet;
    Packet packets[TOTAL_PACKETS];                                      // create array for configured packets

/*--------------------------------------------------------------------------------
 * DDModbusMaster packet and Modbus communications port initialisation:
 */
void mbPacketInit() {
    charge_controller.modbus_construct(&packets[PACKET1], SLAVE_ID, READ_HOLDING_REGISTERS,
            MB_scale_volts_high,
            TOTAL_PACKET_1,
            v_scaling_high);

    charge_controller.modbus_construct(&packets[PACKET2], SLAVE_ID, READ_HOLDING_REGISTERS,
            MB_vBattery_pv,
            TOTAL_PACKET_2,
            vBattery_pv);

    charge_controller.modbus_construct(&packets[PACKET3], SLAVE_ID, READ_HOLDING_REGISTERS,
            MB_vBattery_sp,
            TOTAL_PACKET_3,
            vBattery_sp);

    charge_controller.modbus_configure(MB_SERIAL_PORT, MB_BAUD, SERIAL_8N1, MB_TIMEOUT, MB_POLL_RATE, MB_RETRY, MB_TX_ENABLE_PIN, packets,
  TOTAL_PACKETS, regs, onCallDone);
}
#endif /* CHARGE_CONTROLLER_NUMBER == 20 */

/*--------------------------------------------------------------------------------
 * DDModbusMaster Callback method:
 * Common callback method to handle return of Modbus call.
 * Success and error handling, see ErrorTypes enum in DDModbusMaster.h
 * If no problem occurred errorType will be errSuccess (0).
 */
void onCallDone(Packet *toPacket, errorTypes tErrorType = errSuccess, uint8_t tErrorCode = 0 ){
//  guiPrintMesages("callback");                                        // debug message for checking callback is called:
    switch (tErrorType)
    {
    case errSuccess:                                                    // successful operation:
        switch (toPacket->address)
        {
        case MB_vBattery_pv:
            digitalWrite(LED, !digitalRead(LED));
            if (mb_status.poll_count < 9999)                            // keep track of poll count for this packet:
                mb_status.poll_count++;
            else
            {
                mb_status.poll_count_10K++;
                mb_status.poll_count = 0;
            }
#if (CHARGE_CONTROLLER_TYPE == 0) || (CHARGE_CONTROLLER_TYPE == 10)     // Midnite Solar Classic and Simulator:
            cc_param.vBattery_pv = regs[vBattery_pv];                   // modbus battery volts:
            cc_param.vArray_pv = regs[vArray_pv];                       // modbus pv array volts:
            cc_param.battery_amps = regs[iBattery_pv];                  // modbus battery charge amps:
            cc_param.kwh= regs[kwhr_pv];                                // modbus accumulated kwhr:
            cc_param.watts = regs[watts_pv];                            // modbus battery charge watts:
            cc_param.mode = regs[mode_pv] >> 8;                         // high byte is charge state code:
            cc_param.vBattery_sp = regs[vBattery_sp];                   // modbus battery setpoint
#endif
#if (CHARGE_CONTROLLER_TYPE == 20)                                      // MorningStar TriStar:
            cc_param.vBattery_pv = regs[vBattery_pv] * scale_volts;     // modbus battery volts:
            cc_param.vArray_pv = regs[vArray_pv] * scale_volts;         // modbus pv array volts:
            cc_param.battery_amps = regs[iBattery_pv] * scale_amps;     // modbus battery charge amps:
            cc_param.kwh= regs[kwhr_pv];                                // modbus accumulated kwhr:
            cc_param.watts = regs[watts_pv];                            // modbus battery charge watts:
            cc_param.mode = regs[mode_pv] >> 8;                         // high byte is charge state code:
            cc_param.vBattery_sp = regs[vBattery_sp] * scale_volts;     // modbus battery setpoint
#endif
            reed_param.volt_setpoint = cc_param.vBattery_sp - reed_param.pid_sp_offset; // calculate diversion setpoint:
            reed_param.pid_sp_low_clamp = cc_param.vBattery_sp - LOW_CLAMP_VOLTS_SP;    // calculate diversion low limit clamp:
            guiPrintLive();                                             // on success update live data on GUI:
            break;
        default:
            break;
        }
        break;

    case errTimeOut:                                                    // on timeout increment timeout count:
        mb_status.timeouts++;
        void guiPrintModbusTimeout();
        break;

    case errException:                                                  // on exception error:
        mb_status.address = toPacket->address;
        mb_status.code = tErrorCode;
        void guiPrintModbusException();
        break;

    default:                                                            // on unhandled error:
        mb_status.address = toPacket->address;
        mb_status.code = tErrorCode;
        void guiPrintModbusUnhandled();
        break;
    }
    return;
}

#endif /* modbus_maps_H_ */
```