COFFRET ARDNBUS – FONCTIONNALITES, RACCORDEMENTS ET CONFIGURATION LOGICIELLE.

ⓘ Avant-propos

🔧 **Objectif de ce coffret :**

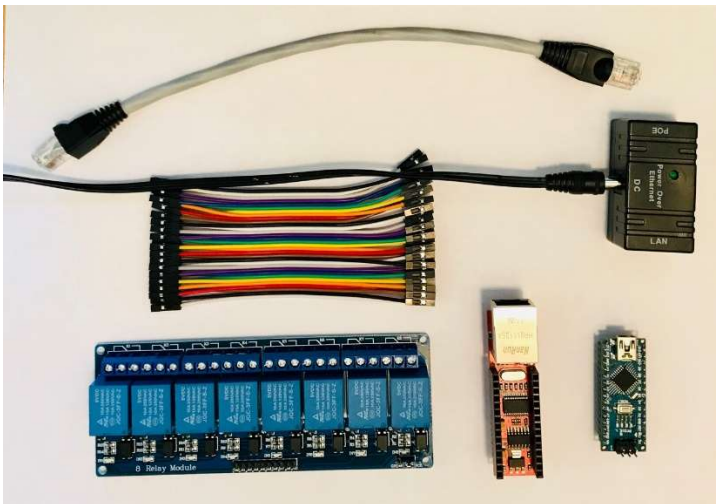
Disposer d'un coffret I/O sous protocole Modbus pour activer ou désactiver dynamiquement les fonctionnalités suivantes :

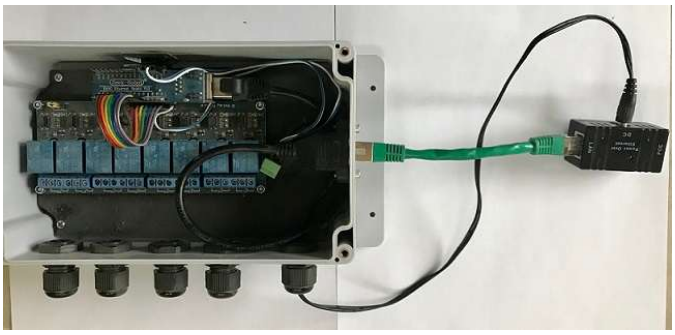
- **8 Relais 1 contact NO et ou NF,**
- **5 Entrées digitales,**
- **4 Sondes DS18B20 sur bus onewire,**
- **1 Sonde DHT22,**
- **1 Entrée PIR.**

Tous les registres sont créés au démarrage du coffret. Pour s'y connecter utiliser Modbus doctor ou tous autres superviseurs. L'IP de base 192.168.1.254.

€ Hardware

- Un Arduino pour moi le nano pour l'encombrement,
- Un Shield Ethernet ENC28J60, pour nano aussi.
- Une carte relais de 2 à 16 relais (Ps sur 16 relais seulement 14 sont utilisables par le nano)
- Des fils DuPont
- Alim 5Vdc 2A et/ou un split pour faux POE 😉
- Un coffret
- Des presses étoupes PG7





🕮 Library

Utilise les Library ModbusIP_ENC28J60. Revue et corrigé pour que le buffer soit défini par le fichier .ino.

ModbusIP_ENC28J60.cpp

```
/*
  ModbusIP_ENC28J60.cpp - Source for Modbus IP ENC28J60 Library
  Copyright (C) 2015 André Sarmento Barbosa
*/

#include "ModbusIP_ENC28J60.h"

// Suppression de la déclaration du buffer ici (doit être dans EtherCard lib)

/*byte Ethernet::buffer[MODBUSIP_MAXFRAME];*/ // <-- supprimé ou commenté

ModbusIP::ModbusIP() {
  ether.hisport = MODBUSIP_PORT;
}

void ModbusIP::config(uint8_t *mac) {
  ether.begin(MODBUSIP_MAXFRAME, mac, ENC28J60_CS);
  ether.dhcpSetup();
}

void ModbusIP::config(uint8_t *mac, uint8_t * ip) {
  ether.begin(MODBUSIP_MAXFRAME, mac, ENC28J60_CS);
  ether.staticSetup(ip);
}

void ModbusIP::config(uint8_t *mac, uint8_t * ip, uint8_t * dns) {
  ether.begin(MODBUSIP_MAXFRAME, mac, ENC28J60_CS);
  ether.staticSetup(ip, 0, dns);
}

void ModbusIP::config(uint8_t *mac, uint8_t * ip, uint8_t * dns, uint8_t * gateway) {
  ether.begin(MODBUSIP_MAXFRAME, mac, ENC28J60_CS);
  ether.staticSetup(ip, gateway, dns);
}

void ModbusIP::config(uint8_t *mac, uint8_t * ip, uint8_t * dns, uint8_t * gateway, uint8_t * subnet) {
  ether.begin(MODBUSIP_MAXFRAME, mac, ENC28J60_CS);
  ether.staticSetup(ip, gateway, dns, subnet);
}

void ModbusIP::task() {
  word len = ether.packetReceive();
  word pos = ether.packetLoop(len);

  if (pos) {
```

```
    int i = 0;
    while (i < 7) {
      _MBAP[i] = Ethernet::buffer[pos+i];
       i++;
    }

    _len = _MBAP[4] << 8 | _MBAP[5];
    _len--; // Do not count with last byte from MBAP
    if (_MBAP[2] !=0 || _MBAP[3] !=0) return;   //Not a MODBUSIP packet
    if (_len > MODBUSIP_MAXFRAME) return;     //Length is over MODBUSIP_MAXFRAME

    _frame = (byte*) malloc(_len);
    i = 0;
    while (i < _len){
      _frame[i] = Ethernet::buffer[pos+7+i];  //Forget MBAP and take just modbus pdu
      i++;
    }

    this->receivePDU(_frame);

    if (_reply != MB_REPLY_OFF) {
      //MBAP
      _MBAP[4] = (_len+1) >> 8;    //_len+1 for last byte from MBAP
      _MBAP[5] = (_len+1) & 0x00FF;

      BufferFiller bfill = ether.tcpOffset();
      bfill.emit_raw((const char *)_MBAP, 7);
      bfill.emit_raw((const char *)_frame, _len);
#ifdef TCP_KEEP_ALIVE
      ether.httpServerReplyAck ();
      ether.httpServerReply_with_flags(bfill.position(), TCP_FLAGS_ACK_V|TCP_FLAGS_PUSH_V);
#else
      ether.httpServerReply(bfill.position());
#endif

    }

    free(_frame);
    _len = 0;
  }
}
```

ModbusIP_ENC28J60.h

```
/*
  ModbusIP_ENC28J60.h - Header for Modbus IP ENC28J60 Library
  Copyright (C) 2015 André Sarmento Barbosa
*/

#ifndef MODBUSIP_ENC28J60_H
#define MODBUSIP_ENC28J60_H

#include <Arduino.h>
#include <Modbus.h>
#include <EtherCard.h>

#define MODBUSIP_PORT      502
#define MODBUSIP_MAXFRAME   700     // Doit être cohérent avec la taille du buffer dans le .cpp et le sketch principal

#define ENC28J60_CS        10      // Pin CS par défaut du ENC28J60 (adapter si besoin)
#define TCP_KEEP_ALIVE              // Option pour garder la connexion TCP active

class ModbusIP : public Modbus {
 private:
  byte _MBAP[7];
  byte* _frame = nullptr;
```

```
  uint16_t _len = 0;
  uint8_t _reply = 0;

 public:
  ModbusIP();
  void config(uint8_t *mac);
  void config(uint8_t *mac, uint8_t * ip);
  void config(uint8_t *mac, uint8_t * ip, uint8_t * dns);
  void config(uint8_t *mac, uint8_t * ip, uint8_t * dns, uint8_t * gateway);
  void config(uint8_t *mac, uint8_t * ip, uint8_t * dns, uint8_t * gateway, uint8_t * subnet);
  void task();
};

#endif // MODBUSIP_ENC28J60_H
```

© Code Source

```
/*
  Library ModbusIP_ENC28J60 - Source for Modbus IP ENC28J60 Library
  Projet : Carte I/O Modbus TCP/IP avec Ethernet ENC28J60
 Fonctions : 8 relais, 5 entrées digitales, 4 sondes DS18B20, DHT22, PIR, watchdog, EEPROM
 Registres Jeedom compatibles + registres de config IP/MAC + registres de modules 920-924 persistants
 Bibliothèque ModbusIP_ENC28j60 reprise pour gestion du buffer.
 Auteur : Fabrice / ChatGPT
*/

#include <EtherCard.h>
#include <ModbusIP_ENC28J60.h>
#include <EEPROM.h>
#include <avr/wdt.h>
#include <DallasTemperature.h>
#include <OneWire.h>
#include <DHT.h>

#define BUFFER_SIZE 700
uint8_t ENC28J60::buffer[BUFFER_SIZE];

#define EEPROM_FLAG_ADDR 0
#define EEPROM_FLAG_VALID 0x42
#define EEPROM_IP_START 1
#define EEPROM_MAC_LAST_BYTE 5

#define EEPROM_FLAGS_START 10 // Adresse EEPROM pour les registres 920 à 924
#define EEPROM_RELAY_ENABLED_ADDR  (EEPROM_FLAGS_START + 0)
#define EEPROM_DI_ENABLED_ADDR     (EEPROM_FLAGS_START + 1)
#define EEPROM_SONDE_ENABLED_ADDR  (EEPROM_FLAGS_START + 2)
#define EEPROM_DHT_ENABLED_ADDR    (EEPROM_FLAGS_START + 3)
#define EEPROM_PIR_ENABLED_ADDR    (EEPROM_FLAGS_START + 4)

#define HOLDING_REG_IP 900
#define HOLDING_REG_MAC_LAST_BYTE 904
//#define HOLDING_REG_FLAGS 905  // PLUS UTILISÉ
#define HOLDING_REG_REBOOT 906
#define HOLDING_REG_SAVE_CONF 907

// Nouveaux registres flags séparés
#define HOLDING_REG_RELAY_ENABLED 920
#define HOLDING_REG_DI_ENABLED 921
#define HOLDING_REG_SONDE_ENABLED 922
#define HOLDING_REG_DHT_ENABLED 923
#define HOLDING_REG_PIR_ENABLED 924

#define RELAIS_ON  LOW   // Active le relais
#define RELAIS_OFF HIGH  // Coupe le relais
#define COIL_FIRST_RELAY 100
#define DISCRETE_INPUT_FIRST_DI 200
#define DISCRETE_INPUT_PIR 506
#define INPUT_REG_FIRST_SONDE 500
#define INPUT_REG_DHT_TEMP 504
#define INPUT_REG_DHT_HUM 505
```

```
#define NBRE_DO 8
#define NBRE_DI 5
#define NBRE_SONDE 4

#define ENC28J60_CS 10

#define DHTPIN A7
#define PIRPIN A5
#define DHTTYPE DHT22
#define ONEWIREPIN A6

int relayPins[NBRE_DO];
int inputPins[NBRE_DI];

OneWire oneWire(ONEWIREPIN);
DallasTemperature sensors(&oneWire);
DHT dht(DHTPIN, DHTTYPE);

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xFE };
byte ip[4];

ModbusIP mb;

word ip_regs_old[4];

// Variables pour anti-écriture EEPROM flags modules au démarrage
bool prevRelay = false;
bool prevDI = false;
bool prevSonde = false;
bool prevDHT = false;
bool prevPIR = false;

bool isFirstBoot = false;

void saveConfigToEEPROM() {
 EEPROM.write(EEPROM_FLAG_ADDR, EEPROM_FLAG_VALID);
 for (int i = 0; i < 4; i++) EEPROM.write(EEPROM_IP_START + i, mb.Hreg(HOLDING_REG_IP + i));
 EEPROM.write(EEPROM_MAC_LAST_BYTE, mb.Hreg(HOLDING_REG_MAC_LAST_BYTE));
 Serial.println(F("💾 Configuration IP/MAC sauvegardée"));
}

bool loadConfigFromEEPROM() {
 if (EEPROM.read(EEPROM_FLAG_ADDR) != EEPROM_FLAG_VALID) {
  Serial.println(F("EEPROM invalide, chargement IP par défaut"));
  ip[0] = 192;
  ip[1] = 168;
  ip[2] = 1;
  ip[3] = 254;
  return false;
 }
 for (int i = 0; i < 4; i++) {
  ip[i] = EEPROM.read(EEPROM_IP_START + i);
 }
 mac[5] = EEPROM.read(EEPROM_MAC_LAST_BYTE);
 Serial.print(F("Config chargée IP: "));
 Serial.print(ip[0]); Serial.print("."); Serial.print(ip[1]); Serial.print("."); Serial.print(ip[2]); Serial.print("."); Serial.println(ip[3]);
 return true;
}

void saveModuleFlagsToEEPROM() {
 for (int i = 0; i < 5; i++) {
  EEPROM.update(EEPROM_FLAGS_START + i, mb.Hreg(HOLDING_REG_RELAY_ENABLED + i));
 }
 Serial.println(F("💾 États modules (920 à 924) sauvegardés dans EEPROM"));
}

void loadModuleFlagsFromEEPROM() {
 for (int i = 0; i < 5; i++) {
  mb.Hreg(HOLDING_REG_RELAY_ENABLED + i, EEPROM.read(EEPROM_FLAGS_START + i));
 }
 Serial.println(F("📦 États modules (920 à 924) restaurés depuis EEPROM"));
}

void printModuleStates() {
```

```
  Serial.println(F("=== États modules ==="));
  Serial.print(F("Relais (reg 920) : ")); Serial.println(mb.Hreg(HOLDING_REG_RELAY_ENABLED) ? "ON" : "OFF");
  Serial.print(F("Entrées digitales (reg 921) : ")); Serial.println(mb.Hreg(HOLDING_REG_DI_ENABLED) ? "ON" : "OFF");
  Serial.print(F("Sondes DS18B20 (reg 922) : ")); Serial.println(mb.Hreg(HOLDING_REG_SONDE_ENABLED) ? "ON" : "OFF");
  Serial.print(F("DHT22 (reg 923) : ")); Serial.println(mb.Hreg(HOLDING_REG_DHT_ENABLED) ? "ON" : "OFF");
  Serial.print(F("PIR (reg 924) : ")); Serial.println(mb.Hreg(HOLDING_REG_PIR_ENABLED) ? "ON" : "OFF");
  Serial.println(F("======================"));
}

void printRelaysState() {
  Serial.println(F("=== État relais ==="));
  for (uint8_t i = 0; i < NBRE_DO; i++) {
    bool state = mb.Coil(COIL_FIRST_RELAY + i);
    Serial.print(F("Relais ")); Serial.print(i+1); Serial.print(F(" : "));
    Serial.println(state ? "ON" : "OFF");
  }
  Serial.println(F("======================"));
}

void printDigitalInputsState() {
  Serial.println(F("=== État entrées digitales ==="));
  for (uint8_t i = 0; i < NBRE_DI; i++) {
    bool state = digitalRead(inputPins[i]);
    Serial.print(F("Entrée digitale ")); Serial.print(i+1); Serial.print(F(" : "));
    Serial.println(state ? "HIGH" : "LOW");
  }
  Serial.println(F("======================"));
}

void printPirState() {
  bool state = digitalRead(PIRPIN);
  Serial.print(F("État PIR : "));
  Serial.println(state ? "Mouvement détecté" : "Aucun mouvement");
  Serial.println(F("======================"));
}

void printSensorValues() {
  Serial.println(F("=== Valeurs sondes ==="));
  for (uint8_t i = 0; i < NBRE_SONDE; i++) {
    int val = mb.Ireg(INPUT_REG_FIRST_SONDE + i);
    float temp = (val / 100.0) - 100.0;
    Serial.print(F("DS18B20 Sonde ")); Serial.print(i+1); Serial.print(F(" : "));
    if (val == 0) Serial.println(F("Erreur / Déconnectée"));
    else Serial.print(temp, 2), Serial.println(F(" °C"));
  }
  int dhtTempVal = mb.Ireg(INPUT_REG_DHT_TEMP);
  int dhtHumVal = mb.Ireg(INPUT_REG_DHT_HUM);
  float dhtTemp = (dhtTempVal / 100.0) - 100.0;
  float dhtHum = dhtHumVal / 100.0;
  Serial.print(F("DHT22 Température : "));
  if (dhtTempVal == 0) Serial.println(F("Erreur"));
  else Serial.print(dhtTemp, 2), Serial.println(F(" °C"));
  Serial.print(F("DHT22 Humidité : "));
  if (dhtHumVal == 0) Serial.println(F("Erreur"));
  else Serial.print(dhtHum, 2), Serial.println(F(" %"));
  Serial.println(F("======================"));
}
  //Config du maintien du PIR
  unsigned long pirLastTriggerTime = 0;
  const unsigned long PIR_HOLD_TIME_MS = 10000;  // 10 secondes de maintien
  bool pirStateHold = false;

void setup() {
  Serial.begin(9600);
  delay(500);

  // Watchdog : lire cause reset + désactiver au début
  uint8_t mcusr = MCUSR;
  MCUSR = 0;
  wdt_disable();

  if (mcusr & _BV(WDRF)) {
    Serial.println(F("\n🟢 Démarrage normal"));
  } else {
    Serial.println(F("\n🟡 Premier démarrage"));
```

```
  isFirstBoot = true;  // Indique 1er boot
 }

 // Charger config IP/MAC
 bool validConfig = loadConfigFromEEPROM();

 // Pins relais : D2 à D9 (8)
for (int i = 0; i < NBRE_DO; i++) {
 relayPins[i] = 2 + i;
 pinMode(relayPins[i], OUTPUT);
 digitalWrite(relayPins[i], RELAIS_OFF);  // tous éteints au démarrage
}
 // Pins entrées digitales : A0 à A4 (5)
 for (int i = 0; i < NBRE_DI; i++) {
  inputPins[i] = A0 + i;
  pinMode(inputPins[i], INPUT);
 }
 pinMode(PIRPIN, INPUT);

 sensors.begin();
 dht.begin();

 // Registres IP/MAC
 for (int i = 0; i < 4; i++) {
  mb.addHreg(HOLDING_REG_IP + i, ip[i]);
  ip_regs_old[i] = ip[i];
 }
 mb.addHreg(HOLDING_REG_MAC_LAST_BYTE, mac[5]);

 // Registres flags séparés 920 à 924, initialisés à 0 (OFF)
 mb.addHreg(HOLDING_REG_RELAY_ENABLED, 0);
 mb.addHreg(HOLDING_REG_DI_ENABLED, 0);
 mb.addHreg(HOLDING_REG_SONDE_ENABLED, 0);
 mb.addHreg(HOLDING_REG_DHT_ENABLED, 0);
 mb.addHreg(HOLDING_REG_PIR_ENABLED, 0);

 mb.addHreg(HOLDING_REG_REBOOT, 0);
 mb.addHreg(HOLDING_REG_SAVE_CONF, 0);

 // Charge les flags modules depuis EEPROM (920 à 924)
 loadModuleFlagsFromEEPROM();

 // Initialiser les variables prev... pour éviter fausse détection au démarrage
 prevRelay = mb.Hreg(HOLDING_REG_RELAY_ENABLED) != 0;
 prevDI    = mb.Hreg(HOLDING_REG_DI_ENABLED) != 0;
 prevSonde = mb.Hreg(HOLDING_REG_SONDE_ENABLED) != 0;
 prevDHT   = mb.Hreg(HOLDING_REG_DHT_ENABLED) != 0;
 prevPIR   = mb.Hreg(HOLDING_REG_PIR_ENABLED) != 0;

 // Coils relais, false au départ
 for (int i = 0; i < NBRE_DO; i++) {
  mb.addCoil(COIL_FIRST_RELAY + i, false);
 }
 // Forcer tous les relais à OFF dans Modbus (évite des activations résiduelles)
 for (int i = 0; i < NBRE_DO; i++) {
  mb.Coil(COIL_FIRST_RELAY + i, false);
 }
 // Discrete inputs DI false au départ
 for (int i = 0; i < NBRE_DI; i++) {
  mb.addIsts(DISCRETE_INPUT_FIRST_DI + i, false);
 }
 // PIR discrete input false
 mb.addIsts(DISCRETE_INPUT_PIR, false);

 // Registres sondes initialisés à 0
 for (int i = 0; i < NBRE_SONDE; i++) {
  mb.addIreg(INPUT_REG_FIRST_SONDE + i, 0);
 }
 mb.addIreg(INPUT_REG_DHT_TEMP, 0);
 mb.addIreg(INPUT_REG_DHT_HUM, 0);

 // Initialisation Ethernet ENC28J60
 if (ether.begin(BUFFER_SIZE, mac, ENC28J60_CS) == 0) {
  Serial.println(F("Erreur init ENC28J60"));
  while (1);
 }
```

```
  ether.staticSetup(ip);
  mb.config(mac, ip);

  Serial.println(F("Démarrage OK."));

  wdt_enable(WDTO_8S);

////////// DEBUG /////////////////
  Serial.println("[DEBUG] État initial des coils :");
for (int i = 0; i < NBRE_DO; i++) {
  Serial.print("Coil ");
  Serial.print(100 + i);
  Serial.print(" = ");
  Serial.println(mb.Coil(100 + i) ? "ON" : "OFF");
}

}

void readAndPublishDS18B20() {
  sensors.requestTemperatures();
  for (int i = 0; i < NBRE_SONDE; i++) {
    float tempC = sensors.getTempCByIndex(i);
    if (tempC == DEVICE_DISCONNECTED_C) {
      mb.Ireg(INPUT_REG_FIRST_SONDE + i, 0);
    } else {
      mb.Ireg(INPUT_REG_FIRST_SONDE + i, int((tempC + 100.0) * 100));
    }
  }
}

void readAndPublishDHT22() {
  float h = dht.readHumidity();
  float t = dht.readTemperature();
  if (isnan(h) || isnan(t)) {
    mb.Ireg(INPUT_REG_DHT_TEMP, 0);
    mb.Ireg(INPUT_REG_DHT_HUM, 0);
  } else {
    mb.Ireg(INPUT_REG_DHT_TEMP, int((t + 100.0) * 100));
    mb.Ireg(INPUT_REG_DHT_HUM, int(h * 100));
  }
}

void loop() {
  mb.task(); // 1er Appel fonction modbus
  wdt_reset(); //Reset du watchdog

  ether.packetLoop(ether.packetReceive());

  // Lecture flags modules séparés
  bool relayEnabled = mb.Hreg(HOLDING_REG_RELAY_ENABLED) != 0;
  bool diEnabled = mb.Hreg(HOLDING_REG_DI_ENABLED) != 0;
  bool sondeEnabled = mb.Hreg(HOLDING_REG_SONDE_ENABLED) != 0;
  bool dhtEnabled = mb.Hreg(HOLDING_REG_DHT_ENABLED) != 0;
  bool pirEnabled = mb.Hreg(HOLDING_REG_PIR_ENABLED) != 0;

  mb.task(); // 2nd Appel fonction modbus

  bool changed = false;

  if (relayEnabled != prevRelay) { changed = true; prevRelay = relayEnabled; }
  if (diEnabled != prevDI) { changed = true; prevDI = diEnabled; }
  if (sondeEnabled != prevSonde) { changed = true; prevSonde = sondeEnabled; }
  if (dhtEnabled != prevDHT) { changed = true; prevDHT = dhtEnabled; }
  if (pirEnabled != prevPIR) { changed = true; prevPIR = pirEnabled; }

  if (changed) {
    Serial.print(F("[INFO] États modules mis à jour: relais="));
    Serial.print(relayEnabled);
    Serial.print(F(", DI=")); Serial.print(diEnabled);
    Serial.print(F(", sonde=")); Serial.print(sondeEnabled);
    Serial.print(F(", DHT=")); Serial.print(dhtEnabled);
    Serial.print(F(", PIR=")); Serial.println(pirEnabled);

    if (!isFirstBoot) {
      saveModuleFlagsToEEPROM();  // 💾 Sauvegarde auto sauf au 1er boot
```

```
  } else {
    Serial.println(F("⏭️ Sauvegarde ignorée (1er boot)"));
    // Après ce premier changement, on force isFirstBoot à false
    isFirstBoot = false;
  }
}

mb.task(); // 3eme Appel fonction modbus

// Gestion relais, toujours exécutée
for (int i = 0; i < NBRE_DO; i++) {
  bool coilState = mb.Coil(COIL_FIRST_RELAY + i);  // suit Jeedom
  digitalWrite(relayPins[i], relayEnabled ? (coilState ? RELAIS_ON : RELAIS_OFF) : RELAIS_OFF);
}
//////////////DEBUG ////////////////// LOG de changement des relais,
for (int i = 0; i < NBRE_DO; i++) {
  static bool prevCoilState[NBRE_DO] = {false};
  bool coilState = mb.Coil(COIL_FIRST_RELAY + i);
  if (coilState != prevCoilState[i]) {
    Serial.print(F("[WRITE] Registre "));
    Serial.print(COIL_FIRST_RELAY + i);
    Serial.print(F(" (coil): "));
    Serial.println(coilState ? "ON" : "OFF");
    prevCoilState[i] = coilState;
  }
}
mb.task(); // 4eme Appel fonction modbus

// Gestion entrées digitales
if (diEnabled) {
  for (int i = 0; i < NBRE_DI; i++) {
    mb.Ists(DISCRETE_INPUT_FIRST_DI + i, digitalRead(inputPins[i]));
  }
} else {
  for (int i = 0; i < NBRE_DI; i++) {
    mb.Ists(DISCRETE_INPUT_FIRST_DI + i, false);
  }
}

// Gestion PIR pendant un timer de 10 sec
  if (pirEnabled) {
  bool pirRaw = digitalRead(PIRPIN);
  if (pirRaw) {
    // Mouvement détecté : on mémorise le temps
    pirLastTriggerTime = millis();
    pirStateHold = true;
  } else {
    // Si pas de détection, on vérifie si on doit encore maintenir l'état HIGH
    if (pirStateHold && (millis() - pirLastTriggerTime > PIR_HOLD_TIME_MS)) {
      // Temps écoulé, on coupe la détection
      pirStateHold = false;
    }
  }
  mb.Ists(DISCRETE_INPUT_PIR, pirStateHold);
} else {
  mb.Ists(DISCRETE_INPUT_PIR, false);
  pirStateHold = false;  // reset si PIR désactivé
}

mb.task(); // 5eme Appel fonction modbus

// Lecture capteurs toutes les 60 sec
static unsigned long lastSensorRead = 0;
if (millis() - lastSensorRead > 60000) {
  lastSensorRead = millis();

  if (sondeEnabled) {
    readAndPublishDS18B20();
  } else {
    for (int i = 0; i < NBRE_SONDE; i++) {
      mb.Ireg(INPUT_REG_FIRST_SONDE + i, 0);
    }
  }

  if (dhtEnabled) {
```

```
    readAndPublishDHT22();
   } else {
    mb.Ireg(INPUT_REG_DHT_TEMP, 0);
    mb.Ireg(INPUT_REG_DHT_HUM, 0);
   }
  }

mb.task(); // 6eme Appel fonction modbus

  // Gestion sauvegarde config via registre 907
  static word prev_save = 0;
  word current_save = mb.Hreg(HOLDING_REG_SAVE_CONF);
  if (current_save == 1 && prev_save == 0) {
   Serial.println(F("💾 Déclenchement sauvegarde via registre 907"));
   saveConfigToEEPROM();
   mb.Hreg(HOLDING_REG_SAVE_CONF, 0);

   Serial.println(F("Reboot après sauvegarde..."));
   Serial.flush();
   delay(100);
   wdt_enable(WDTO_15MS);
   while (1) {}
  }
  prev_save = current_save;

  // Gestion reboot via registre 906
  static word prev_reboot = 0;
  word current_reboot = mb.Hreg(HOLDING_REG_REBOOT);
  if (current_reboot == 1 && prev_reboot == 0) {
   Serial.println(F("Reboot demandé..."));
   Serial.flush();
   delay(100);
   wdt_enable(WDTO_15MS);
   while (1) {}
  }
  prev_reboot = current_reboot;

  // Affichage status périodique
  static unsigned long lastStatusPrint = 0;
  if (millis() - lastStatusPrint > 20000) {
   lastStatusPrint = millis();
   printModuleStates();
   if (relayEnabled) printRelaysState();
   if (diEnabled) printDigitalInputsState();
   if (pirEnabled) printPirState();
   if (sondeEnabled || dhtEnabled) printSensorValues();
  }

  mb.task(); // 7eme Appel fonction modbus

  // Lecture commandes série
  checkSerialCommand();
}

void checkSerialCommand() {
  if (Serial.available()) {
   String cmd = Serial.readStringUntil('\n');
   cmd.trim();

   if (cmd.equalsIgnoreCase("reboot")) {
    Serial.println(F("♻️ Reboot via console..."));
    Serial.flush();
    delay(100);
    wdt_enable(WDTO_15MS);
    while (1);
   }

   else if (cmd.equalsIgnoreCase("conf")) {
    printModuleStates();
    printRelaysState();
    printDigitalInputsState();
    printPirState();
    printSensorValues();
   }
```

```
  else if (cmd.equalsIgnoreCase("reset")) {
    Serial.println(F(" 🔁 Réinitialisation IP/MAC par défaut..."));

    byte new_ip[] = {192, 168, 1, 254};
    for (int i = 0; i < 4; i++) {
      ip[i] = new_ip[i];
      mb.Hreg(HOLDING_REG_IP + i, ip[i]);
    }

    mac[5] = 0xFE;
    mb.Hreg(HOLDING_REG_MAC_LAST_BYTE, mac[5]);

    saveConfigToEEPROM();

    Serial.println(F(" ✅ Configuration par défaut appliquée et sauvegardée. Reboot..."));
    Serial.flush();
    delay(100);
    wdt_enable(WDTO_15MS);
    while (1);
  }

  else if (cmd.equalsIgnoreCase("etat")) {
    Serial.println(F(" 📊 État du système :"));
    printModuleStates();
    uint16_t relay = mb.Hreg(HOLDING_REG_RELAY_ENABLED);
    uint16_t di = mb.Hreg(HOLDING_REG_DI_ENABLED);
    uint16_t pir = mb.Hreg(HOLDING_REG_PIR_ENABLED);
    uint16_t sonde = mb.Hreg(HOLDING_REG_SONDE_ENABLED);
    uint16_t dht = mb.Hreg(HOLDING_REG_DHT_ENABLED);

    if (relay) printRelaysState();
    if (di) printDigitalInputsState();
    if (pir) printPirState();
    if (sonde || dht) printSensorValues();
  }

  else {
    Serial.print(F("Commande inconnue : "));
    Serial.println(cmd);
  }
 }
}
```

⚙️ Registres 100 à 107 – Relais I/O TCP

(🔹 Coils (Bobines) : Relais de puissance)

Adresse	Fonction	Valeur	Adresse physique I/O
100	Relais 1	0 ou 1	Pin D2
101	Relais 2	0 ou1	Pin D3
102	Relais 3	0 ou1	Pin D4
103	Relais 4	0 ou1	Pin D5
104	Relais 5	0 ou1	Pin D6
105	Relais 6	0 ou1	Pin D7
106	Relais 7	0 ou1	Pin D8

107	Relais 8	0 ou1	Pin D9

Registres 200 à 204 + 506 – Entrées digitales et PIR

(◆ Discrete Inputs : Entrées digitales (interrupteurs)

Adresse	Fonction	Adresse physique I/O
200	Entrée digitale 1	Pin A0 (Digitale input)
201	Entrée digitale 2	Pin A1
202	Entrée digitale 3	Pin A2
203	Entrée digitale 4	Pin A3
204	Entrée digitale 5	Pin A4
506	Entrée PIR (mouvement)	Pin A5

Registres 500 à 505 – Sondes et DHT

◆ Input Registers : Températures / capteurs analogiques (lecture uniquement)

La relève des sondes s'effectue toutes les 60 secondes (timer fixé dans le code)

Adresse	Fonction	Format	Adresse physique I/O
500	Temp DS18B20 - sonde 1	(temp+100)*100	Pin **A6 (bus 1 Wire)**
501	Temp DS18B20 - sonde 2	(temp+100)*100	Pin **A6 (bus 1 Wire)**
502	Temp DS18B20 – sonde 3	(temp+100)*100	Pin **A6 (bus 1 Wire)**
503	Temp DS18B20 - sonde 4	(temp+100)*100	Pin **A6 (bus 1 Wire)**
504	Température DHT22	(temp+100)*100	Pin **A7**
505	Humidité DHT22	en % * 100	Pin **A7**

Registres de configuration 900 à 907

Fonction	Registre Holding	Valeur
Adresse IP	900 à 903	Octets IP
MAC dernier octet	904	Byte MAC[5]
Reboot	906	1 = redémarrage
Save eeprom	907	1 = Save eeprom + reboot

Registre 900 à 903 – Configuration de l'adresse IP.

- Par défaut l'ip du coffret est fixé à : **192.168.1.254**

Adresse	Fonction	Valeur par défaut
900	Premier bloc IP	192
901	Second bloc IP	168
902	Troisième bloc IP	1
903	Quatrième bloc IP	254

Une fois connecté au coffret, attribuer une IP fixe à distance.

Dès qu'une valeur est envoyer alors la carte rebootera automatiquement pour prise en compte. (voir WATCHDOG)

Registre 904 – Configuration de l'adresse MAC.

- Par défaut l'adresse MAC est : DE:AD:BE:EF:FE:FE

Ce registre permet modifier le 6ème et dernier octet de l'adresse MAC du coffret afin qu'il soit unique. Obligatoire pour un bon fonctionnement multi-coffret.

Ci-dessous 10 adresses possibles selon la valeur du registre :

Registre HR 904	Adresse MAC complète
1	DE:AD:BE:EF:FE:**01**
2	DE:AD:BE:EF:FE:**02**
10	DE:AD:BE:EF:FE:**0A**
16	DE:AD:BE:EF:FE:**10**

100	DE:AD:BE:EF:FE:**64**
123	DE:AD:BE:EF:FE:**7B**
200	DE:AD:BE:EF:FE:**C8**
225	DE:AD:BE:EF:FE:**E1**
254	DE:AD:BE:EF:FE:**FE**
255	DE:AD:BE:EF:FE:**FF**

Dès qu'une valeur est envoyer il faut utiliser le registre 907 pour prise en compte.

Registre 906 – Commande de reboot

- Écrire la valeur 1 déclenche un redémarrage logiciel immédiat.
- Toute autre valeur est ignorée.
- Le registre est automatiquement remis à 0 après traitement.

Registre 907 – Sauvegarde config IP/MAC

Sauvegarde la configuration actuelle (IP et dernier octet MAC) depuis Modbus

- **Type** : Holding Register
- **Adresse** : 907
- **Utilisation** :
  - Après modification des registres 900–903 (IP) et/ou 904 (MAC[5]), écrire : HR[907] = 1
  - La carte sauvegarde la config dans l'EEPROM puis remet HR[907] à 0.
- **Effet dans la console série** :
  - 💾 Déclenchement sauvegarde via registre 907
  - 💾 Configuration IP/MAC sauvegardée

Registres de contrôle des modules (920 à 924)

Adresse registre	Nom	Description	Valeurs possibles
920	HOLDING_REG_RELAY_ENABLED	Active/désactive la gestion des **relais** (Coils 100–107 et Hregs 910–917)	0 = désactivé 1 = activé
921	HOLDING_REG_DI_ENABLED	Active/désactive la lecture des **entrées digitales** (DI A0–A4)	0 = désactivé 1 = activé
922	HOLDING_REG_SONDE_ENABLED	Active/désactive les **sondes DS18B20** (4 sondes)	0 = désactivé 1 = activé
923	HOLDING_REG_DHT_ENABLED	Active/désactive le **capteur DHT22** (température/humidité)	0 = désactivé 1 = activé

Adresse registre	Nom	Description	Valeurs possibles
924	HOLDING_REG_PIR_ENABLED	Active/désactive la lecture du capteur **PIR** (détection mouvement)	0 = désactivé 1 = activé

🎯 **Fonctionnement**

- Ces registres sont des **Holding Registers** Modbus (écriture 16 bits).
- Ils sont **persistants** : toute modification est automatiquement sauvegardée en **EEPROM**.
- Ils sont **chargés au démarrage** et restaurent automatiquement l'état de la configuration.

🧠 **Utilisation**

- Si HOLDING_REG_RELAY_ENABLED = 1, alors les **relais** peuvent être commandés via Jeedom.
- Si = 0, toutes les sorties relais sont **forcées à OFF**, et **aucune commande** ne les activera

🕒 Watchdog (Surveillance)

Le **watchdog** est un minuteur de sécurité matériel. Si le programme ne le "relance" pas régulièrement, il redémarre automatiquement la carte. Cela évite les blocages (par exemple si le réseau se fige). Une sécurité en début de code désactive le watchdog afin d'éviter le brick de la carte.

Comportement dans le code :

- Activé avec wdt_enable(WDTO_8S) : le watchdog déclenche un reset si aucun wdt_reset() n'est appelé dans un délai de 8 secondes.
- Dans loop(), un appel à wdt_reset() permet de le maintenir.
- Lors d'une demande de reboot (registre 906), on force un reboot en activant wdt_enable(WDTO_15MS) puis une boucle infinie while(1) {}.

💾 EEPROM (Mémoire non volatile)

L'**EEPROM** est utilisée pour conserver la configuration IP et MAC même après coupure de courant.

Fonctionnement dans le code :

- Au démarrage :
  - Si EEPROM.read(0) != 0x42, une config par défaut est appliquée.
  - Sinon, les valeurs IP (octets 1à4) et MAC (octets 5à10) sont chargées.
- En fonctionnement :

- Si l'utilisateur modifie l'IP ou le MAC via Jeedom (registres 900à904), la fonction checkAndSaveConfig() détecte les changements et les enregistre avec saveConfigToEEPROM().
- Pour appliquer la nouvelle config il faudra écrire le HR907 à 1

Structure EEPROM :

Adresse EEPROM	Contenu. Un contrôle du premier octet de l'ip est réalisé au démarrage pour éviter toutes erreur de lecture
0	0x42 (valide ?)
1-4	IP (octets 1-4)
5-10	MAC (octets 0-5)

🖧 Commande série

Trois fonctions embarquées sont utilisables par la console série une fois le coffret connecté à un PC + Arduino IDE.

- Conf = Statut de la configuration du coffret
- Reboot = force le reboot sofware du coffret
- Reset = Réinitialise le coffret à ses paramètres par défaut.

📋 Commande conf (console série)

🔍 Objectif :
Afficher en temps réel la configuration courante de la carte I/O Modbus TCP/IP via le port série USB.

🖥️ Utilisation :
Dans le moniteur série (9600 bauds), tapez simplement : conf

📩 Résultat :
=== Configuration ===
IP: 192.168.1.254
MAC: DE:AD:BE:EF:FE:FE
Relais: OFF
DHT22: OFF
PIR: OFF
Dallas: OFF
DI: OFF
==========
IP : adresse IP actuelle (à jour avec les registres HR 900–903)
MAC : adresse MAC complète (les 5 premiers octets sont fixes, le dernier est modifiable via HR 904)

Relais, DHT22, PIR, Dallas, DI : placeholders actuels (affichés "OFF" car non encore dynamiquement liés au hardware)

🔧 Remarques :
Les valeurs affichées reflètent les valeurs en mémoire à l'instant T.
Pour refléter des changements, modifier les registres puis taper conf à nouveau.
La commande est non destructive et peut être appelée à tout moment

🔁 Reboot et Réinitialisation – Carte I/O Modbus TCP/IP (Redémarrage logiciel)

Méthodes disponibles :

1. **Par registre Modbus :**
   - Écrire 1 dans le registre **906** (HR 906).
   - Effet : déclenche un redémarrage logiciel via le Watchdog matériel.
   - Le registre est automatiquement remis à 0.
2. **Par console série (via USB) :**
   - Dans le moniteur série (9600 bauds), envoyer la commande texte : reboot

🔃 Réinitialisation de l'adresse IP et MAC (valeurs par défaut)

Objectif : remettre l'IP à 192.168.1.254 et la MAC à DE:AD:BE:EF:FE:FE
Méthode : console série uniquement

- Dans le moniteur série (9600 bauds), envoyer la commande : reset
- La configuration par défaut est sauvegardée en EEPROM **et un reboot automatique est déclenché**.