BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC CẦN THƠ
TRƯỜNG BÁCH KHOA
KHOA TỰ ĐỘNG HÓA

--ଓ📖ଌ--

BÁO CÁO ĐỒ ÁN

HỌC PHẦN ĐO LƯỜNG VÀ ĐIỀU KHIỂN BẰNG MÁY TÍNH

XÂY DỰNG MÔ HÌNH ĐIỀU KHIỂN BIẾN TẦN

CÁN BỘ HƯỚNG DẪN
ThS.Nguyễn Khắc Nguyên

SINH VIÊN THỰC HIỆN
Nhóm 09
Nguyễn Duy Tân B2110310
Đặng Hoàng Khải B2103972
Trương Duy Linh B2110304
Nguyễn Ngọc Tính B2110316

Cần Thơ, 2025

MỤC LỤC

1. GIỚI THIỆU CHUNG

1.1. Mục tiêu của đề tài

- Tìm hiểu truyền thông Modbus RTU và OPC Server.
- Tìm hiểu về biến tần.
- Kết nối Arduino với OPC Server, điều khiển biến tần trên giao diện Web. Có thể mở rộng quy mô điều khiển trên nhiều thiết bị khác nhau.

1.2. Các tính năng cơ bản

Xây dựng mô hình điều khiển biến tần với các chức năng như sau:

- Cài đặt giá trị về ID trong giao thức truyền thông Modbus RTU cho thiết bị điều khiển và biến tần.
- Có thể bật/tắt biến tần.
- Điều chỉnh chiều quay của của biến tần.
- Điều chỉnh tần số của biến tần.
- Đọc các thông số biến tần gửi về hệ thống điều khiển như: tần số thực tế, giá trị dòng tải và moment xoắn của động cơ.
- Có thể cài đặt trạng thái, tần số hoạt động trong khoảng thời gian nhất định.
- Điều khiển qua nhiều thiết bị được kết nối trong mạng cục bộ.

1.3. Phương pháp thực hiện

Xác định yêu cầu:

- Ghi dữ liệu vào thanh ghi ***Control word*** và ***Frequency reference via the bus (signed value)*** (***8501*** và ***8052***) của biến tần để thực hiện tắt/mở biến tần và điều chỉnh tần số. Đọc được dữ liệu từ các thanh ghi ***Output frequency applied to the motor (signed value)***, ***Current in the motor***, ***Motor torque*** (***3202***, ***3204***, ***3205***) nhằm xác định tần số thực tế, dòng tải và moment xoắn của động cơ. Các thanh ghi này được trình bày trong dataset của biến tần ATV312 [1] (hình 1).

Modbus address	CANopen address	Code	Read/ Write	Name/Descriptio
8501	2037 / 2	CMD	R/W	**Control word** bit 0: "Switch on": active at 1 bit 1: "Disable Voltage": active at 0 bit 2: "Quick Stop": active at 0 bit 3: "Enable Operation": active at 1 bits 4 to 6: Reserved: set to 0 bit 7: Fault state reset: active on rising edge 0 bits 8 to 10: Reserved: set to 0
8502	2037 / 3	LFr	R/W	**Frequency reference via the bus (signed value)** Unit: • 1 = 0.1 Hz if bit 9 of CMI (page 20) = 0 • 1 = 0.018 Hz (resolution 32767 points = 600 Hz)

3202	2002 / 3	rFr	R	**Output frequency applied to the motor (signed value)** Unit: • 1 = 0.1 Hz if bit 9 of CMI (page 20) = 0 • 1 = 0.018 Hz (resolution 32767 points = 600 Hz) if bit 9
3203	2002 / 4	FrH	R	**Frequency reference before ramp (absolute value)** Unit: 0.1 Hz
3204	2002 / 5	LCr	R	**Current in the motor** Unit: 0.1 A
3205	2002 / 6	Otr	R	**Motor torque** Unit: 1% 100% = Nominal motor torque, calculated using the config

Hình 1 Các thanh ghi trong biến tần ATV312

- Điều khiển thông qua giao diện Web.
- Hiển thị các giá trị đọc được lên LCD và giao diện Web.

Lập kết hoạch thiết kế: Sử dụng Arduino làm bộ xử lí chính, sử dụng module ***TTL to RS485 v2*** với chức năng thực hiện truyền thông Modbus giữa vi điều khiển với biến tần và PC.

Thiết kế mạch điện: Thiết kế theo từng khối chức năng:

- Khối nguồn: Sử dụng ***IC 7805*** hạ áp từ 9VDC thành 5VDC.
- Khối hiển thị: Sử dụng ***LCD 16x2***.
- Khối truyền thông: Sử dụng 2 module ***TTL to RS485 v2.***
- Khối điều chỉnh: Sử dụng module ***Rotary Encoder***.
- Khối xử lí trung tâm: Sử dụng module ***Arduino nano***.

Lập trình: Viết chương trình theo từng module tương tứng với phần cứng. Sử dụng Node-RED để tạo giao diện điều khiển trên PC.

Kiểm tra và hiệu chỉnh:

- Kiểm tra mạch theo từng khối đã thiết kế.
- Kiểm tra từng đoạn chương trình với các khối tương ứng.
- Hiệu chỉnh các thông số nhằm cải thiện hiệu suất và độ ổn định.

Thử nghiệm:

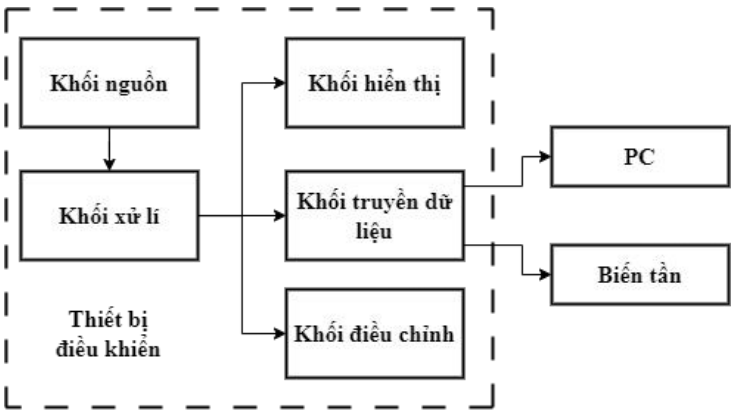
- Thử nghiệm trong thời gian lâu dài, thực hiện nhiều chức năng đồng thời để đảm bảo tính ổn định của thiết bị.
- Sửa lỗi và điều chỉnh khi cần thiết.

2. THỰC HIỆN

2.1. Mô hình hệ thống

Mô tả sơ lược về hệ thống: hệ thống sẽ kết nối với biến tần thông qua cổng RJ45, kết nối với PC thông qua cổng USB. Giao diện điều khiển được thiết kế từ Node-RED sẽ được kết nối với OPC Server, với mục đích điều khiển các ***Tabs***. Dữ liệu trong các ***Tabs*** này sẽ được truyền cho vi điều khiển. Tương ứng với các lệnh nhận được, vi điều khiển sẽ chuyển trạng thái của biến tần theo ý muốn của người điều khiển.

Mô hình hóa hệ thống dạng khối:



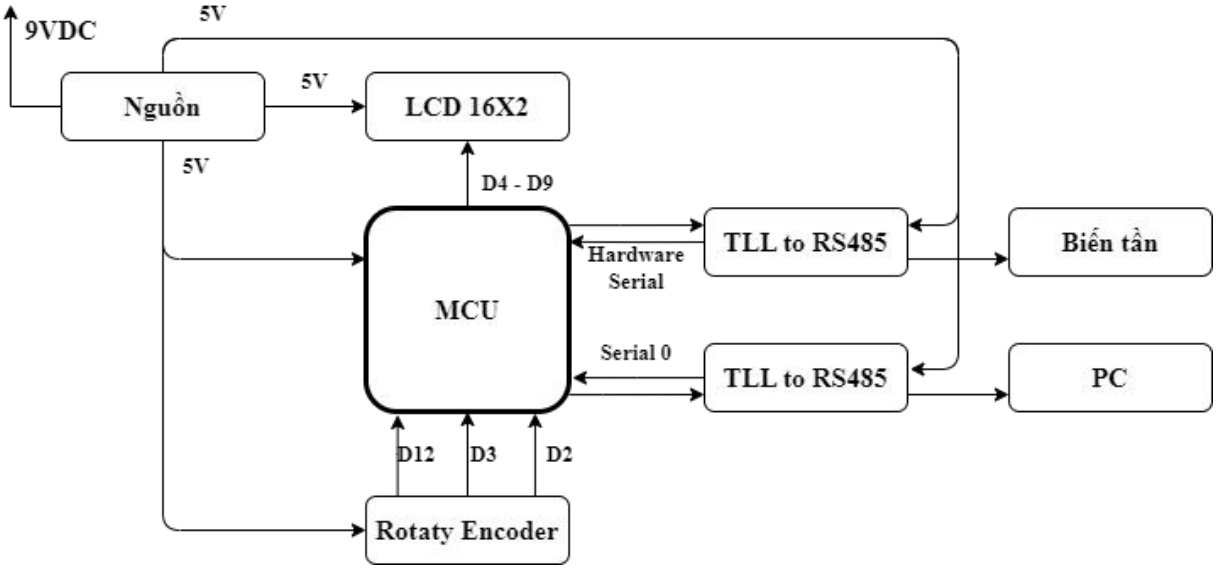
Hình 2 Mô hình hóa hệ thống dạng khối

2.2. Thiết kế phần cứng mạch điện

Mô tả sơ lược hoạt động của mạch điện:

- Hệ thống có 2 chế độ hoạt động: chế độ SETUP VÀ RUN.
- Trong chế độ SETUP, vi điều khiển nhận tín hiệu điều khiển từ Rotary Encoder để điều chỉnh giá trị về ID của thiết bị và biến tần. Các giá trị này được lưu trong bộ nhớ EEPROM của vi điều khiển.
- Trong chế độ RUN, vi điều khiển thu thập dữ liệu từ biến tần, hiển thị các giá trị nhận được lên PC và LCD. Đồng thời nhận lệnh điều khiển từ PC.

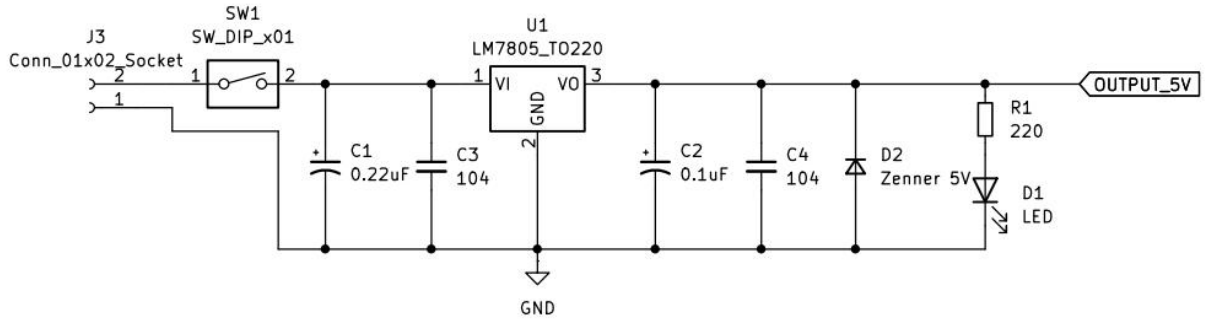
Sơ đồ khối phần cứng mạch điện:



Hình 3 Sơ đồ phần cứng mạch điện

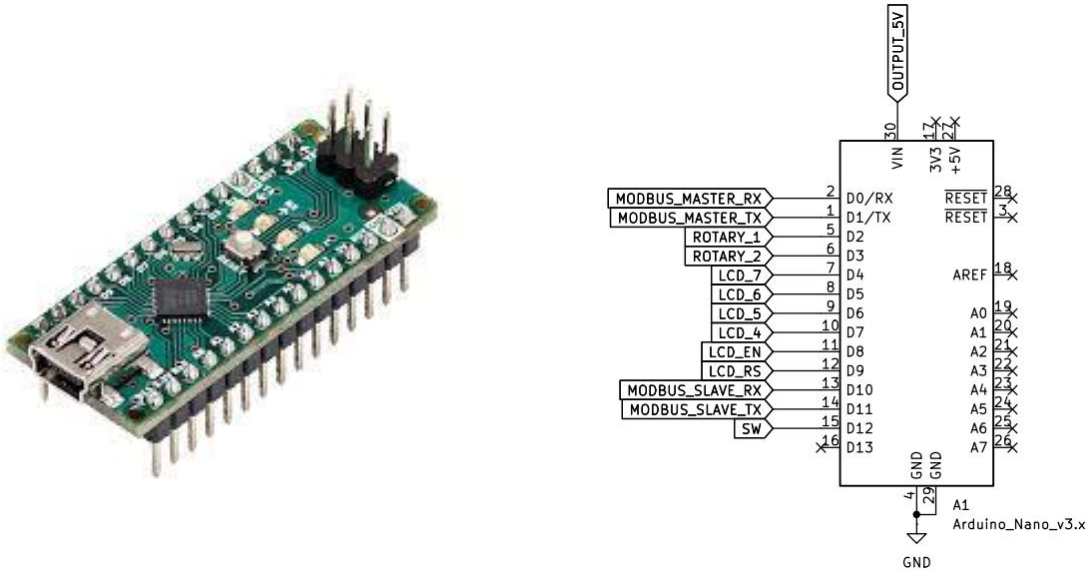
Thiết kế chi tiết mạch nhúng:

- **Khối nguồn:**
  + Chức năng: cung cấp nguồn điện cho toàn bộ thiết bị.
  + Sơ đồ nguyên lý:



Hình 4 Sơ đồ nguyên lý khối nguồn

- **Khối xử lí trung tâm:**
  + Chức năng: sử dụng vi điều khiển Arduino nano để nhận dữ liệu từ PC và module Rotary Encoder, điều khiển và nhận các giá trị phản hồi từ biến tần.
  + Sơ đồ nguyên lý:

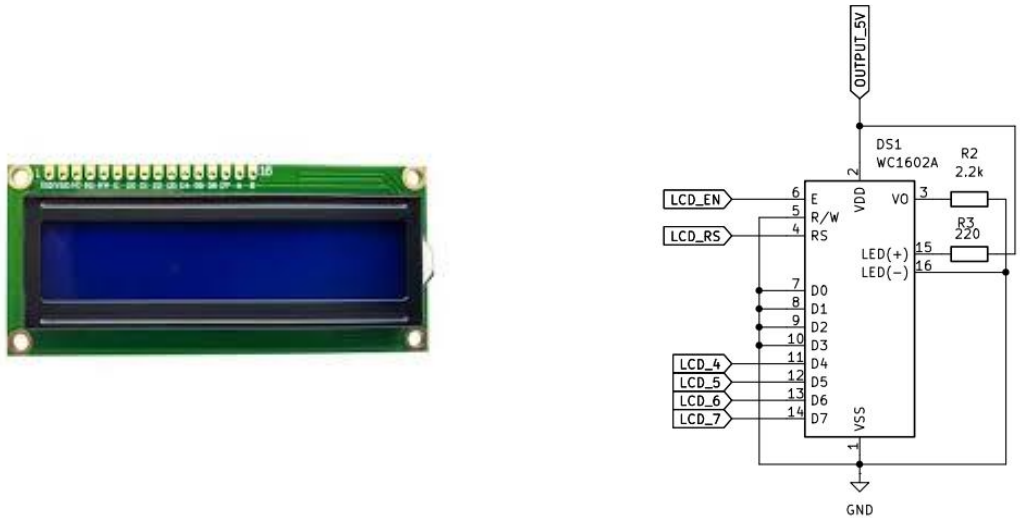


Hình 5 Sơ đồ nguyên lý khối xử lý trung tâm

- **Khối điều chỉnh - cài đặt ID:**
  + Chức năng: cài đặt tần số ID cho thiết bị và biến tần trong giao thức truyền thông Modbus RTU.
  + Sơ đồ nguyên lý:

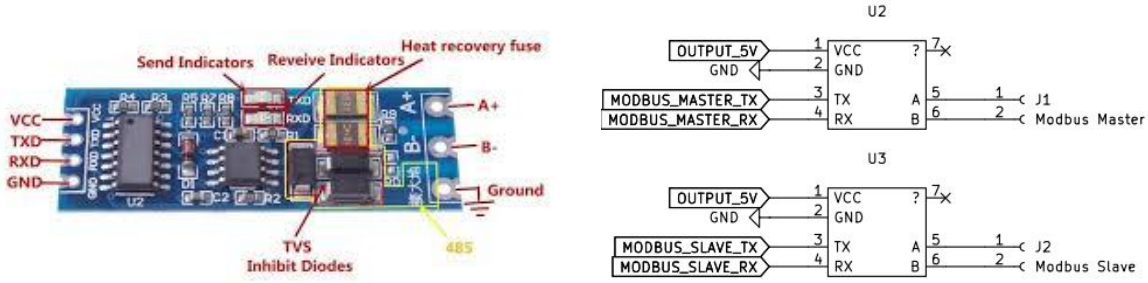
Hình 6 Sơ đồ nguyên lý khối điều chỉnh

- **Khối hiển thị:**
  + Chức năng: trong chế độ SETUP hiển thị giá trị ID cài đặt, trong chế độ RUN hiển thị giá trị tần số và dòng tải của biến tần.
  + Sơ đồ nguyên lý:



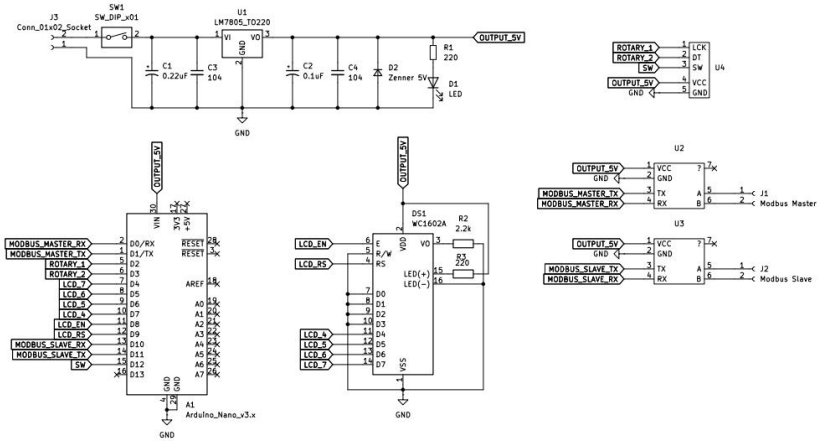
Hình 7 Sơ đồ nguyên lý khối hiển thị

- **Khối truyền dữ liệu:**
  + Chức năng: sử dụng module chuyển đổi tín hiệu TTL sang RS485, giúp truyền dữ liệu đi xa hơn, hạn chế mất dữ liệu và phù hợp với chuẩn truyền thông Modbus.
  + Sơ đồ nguyên lý:



Hình 8 Sơ đồ nguyên lý khối truyền dữ liệu

Sơ đồ nguyên lý tổng quát:



Hình 9 Sơ đồ nguyên lý tổng quát

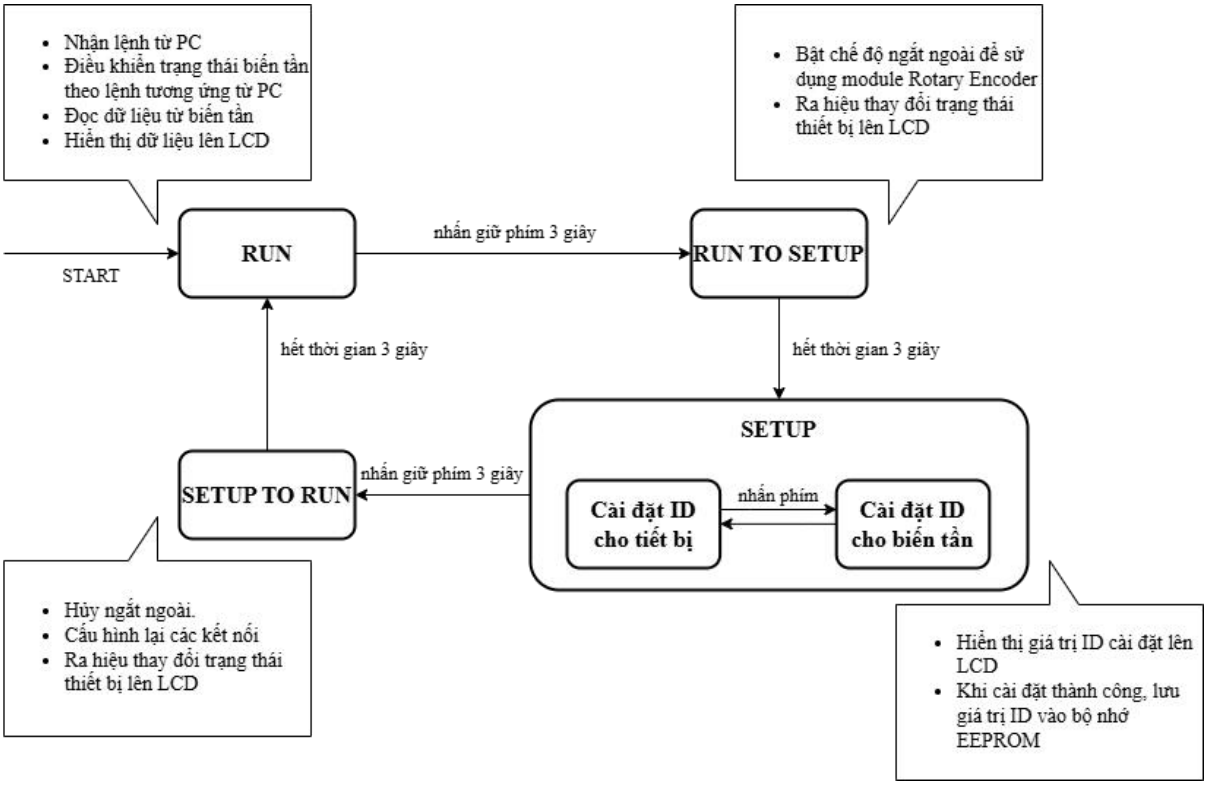
2.3. Thiết kế chương trình điều khiển

2.3.1. Chương trình điều khiển trên board nhúng

Mô tả chức năng chính:

- Khai báo các kết nối đến các GPIO của vi điều khiển, cấu hình cho các ngoại vi như LCD, Rotary Encoder, TTL to RS485.
- Thay đổi chế độ hoạt động giữa SETUP và RUN khi nhấn giữ phím ấn 3 giây trên module Rotary Encoder. Điều chỉnh giá trị ID của thiết bị và biến tần bằng cách xoay núm vặn trên module. Khi thiết lập thành công, giá trị này sẽ được lưu vào bộ nhớ EEPROM của vi điều khiển, sau đó bắt đầu cấu hình lại các cổng kết nối.
- Trong giao tiếp với PC, cấu hình vi điều khiển làm Slave, khai báo 1 thanh ghi dữ liệu để PC có thể tương tác với vi điều khiển.
- Trong giao tiếp với biến tần, cấu hình vi điều khiển làm Master, để có thể ghi và đọc dữ liệu từ các thanh ghi trong biến tần. Dữ liệu nhận được sẽ được truyền vào thanh ghi bên trong vi điều khiển.
- In giá trị đọc được lên màng hình LCD 16x2.

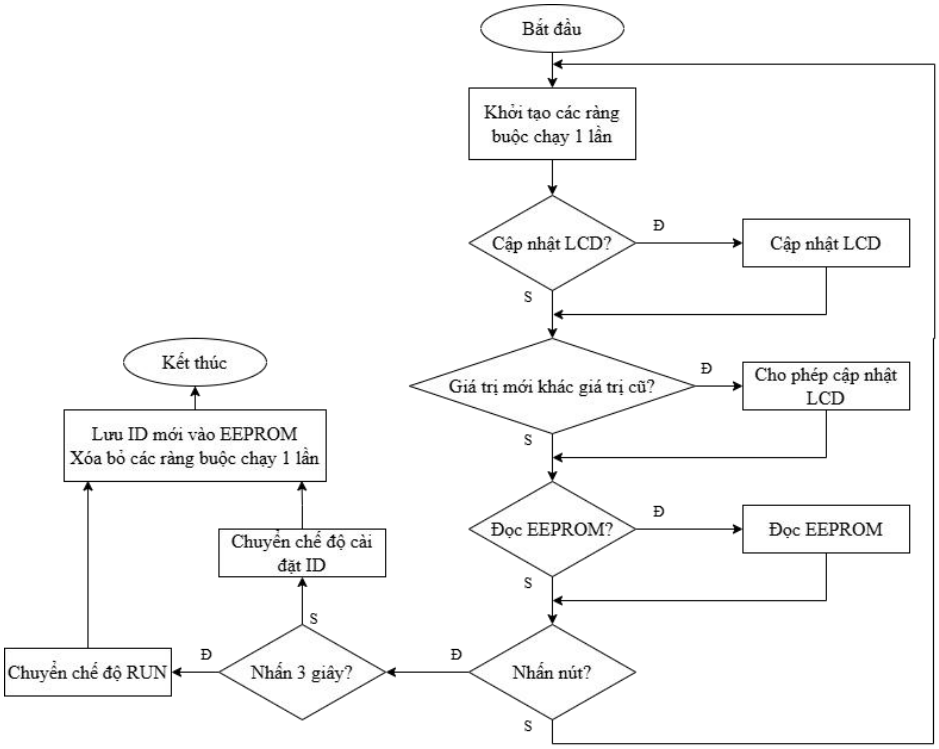
Sơ đồ trạng thái của thiết bị:



Hình 10 Sơ đồ trạng thái của thiết bị

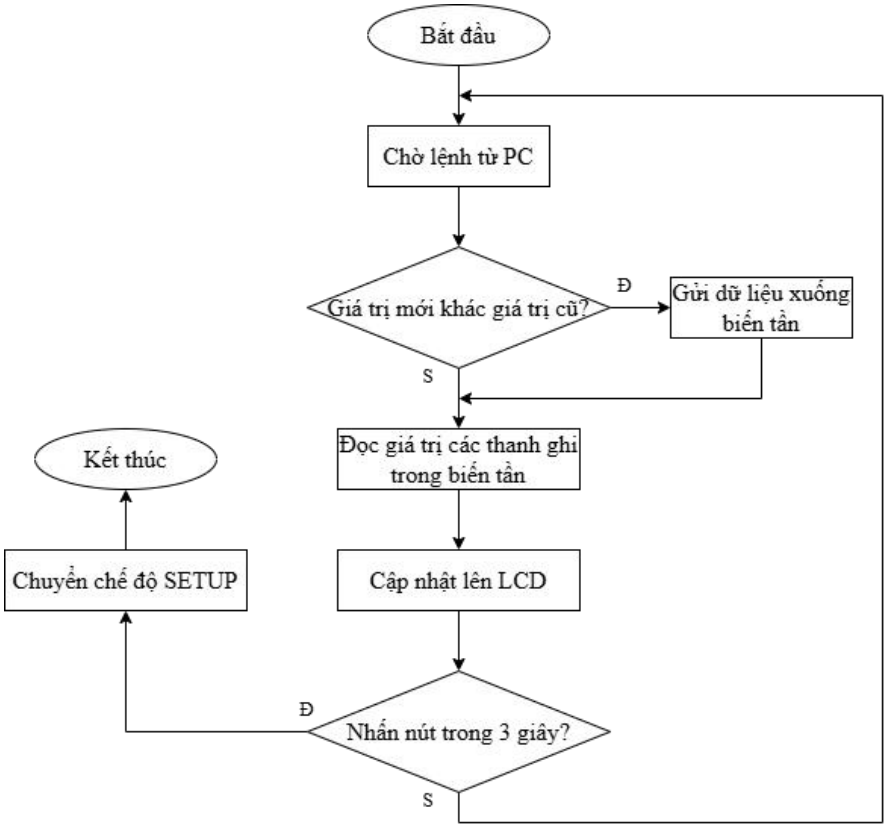
Lưu đồ giải thuật của các chương trình con chính yếu:

Lưu đồ giải thuật chương trình SETUP:



Hình 11 Lưu đồ giải thuật chức năng SETUP

Lưu đồ giải thuật chương trình RUN:



Hình 12 Lưu đồ giải thuật chức năng RUN

Đoạn code chương trình:

```
#include <SoftwareSerial.h>
#include <ModbusMaster.h>
#include <ModbusRTUSlave.h>
#include <LiquidCrystal.h>
#include <EEPROM.h>
//--------------------------------------------------
/**
  * Khai báo nút nhấn và module rotary encoder
  */
#define SETUP_PIN 12
#define PHA1_PIN 2
#define PHA2_PIN 3

//--------------------------------------------------
/**
  * Khai báo địa chỉ EEPROM lưu ID của thiết bị và biến tần.
  * ID của thiết bị được lưu vào -- 0
  * ID của biến tần được lưu vào -- 1
  */
#define DEVICE_ID_ADDRESS_IN_EEPROM 0
#define INVERTER_ID_ADDRESS_IN_EEPROM 1


//--------------------------------------------------
/**
  * Thiết lập lcd 16x2
  */
#define RS_PIN 9
#define EN_PIN 8
#define D4_PIN 7
#define D5_PIN 6
#define D6_PIN 5
#define D7_PIN 4
LiquidCrystal lcd(RS_PIN, EN_PIN, D4_PIN, D5_PIN, D6_PIN, D7_PIN);

//--------------------------------------------------
/**
  * Thiết lập cổng Serial phụ
  * Do Arduino nano chỉ hỗ trợ 1 cổng serial
  * Rx = 10, Tx = 11
  */
SoftwareSerial MODBUS_MASTER_SERIAL(10, 11);

//--------------------------------------------------
```

```
/**
  * Thiết lập thông số Modbus Master - Điều khiển biến tần
  * mbm1 điều khiển biến tần
  */
#define MODBUS_MASTER_BAUD 9600
#define MODBUS_MASTER_CONFIG SERIAL_8N1
#define MODBUS_INVERTER_SLAVE_ID EEPROM.read(INVERTER_ID_ADDRESS_IN_EEPROM)
ModbusMaster mbm1;

// Thiết lập thông số Modbus Slave - Nhận lệnh điều khiển từ PC
#define MODBUS_SLAVE_SERIAL Serial
#define MODBUS_SLAVE_BAUD 9600
#define MODBUS_SLAVE_CONFIG SERIAL_8N1
#define MODBUS_SLAVE_INIT_ID EEPROM.read(DEVICE_ID_ADDRESS_IN_EEPROM)
ModbusRTUSlave mbs(MODBUS_SLAVE_SERIAL);

//-----------------------------------------------------
/**
  * Địa chỉ Modbus điều khiển biến tần
  */
#define CONTROL_WORD 8501
#define FREQUENCY_REFERENCE 8502
#define OUTPUT_FREQUENCY 3202
#define CURRENT_IN_THE_MOTOR 3204
#define MOTOR_TORQUE 3205

// Cài đặt các trạng thái điều khiển trong thanh ghi CONTROL_WORD
#define KHOI_DONG 0x06
#define DUNG 0x100F
#define QUAY_THUAN 0x0F
#define QUAY_NGHICH 0x80F

//-----------------------------------------------------
/**
  * Các thanh ghi trong MCU
  */
typedef enum {
  iCONTROL_WORD,
  iFREQUENCY_REFERENCE,
  iOUTPUT_FREQUENCY,
  iCURRENT_IN_THE_MOTOR,
  iMOTOR_TORQUE,
  iLENGTH_REGISTER
};
uint16_t registerAddress[iLENGTH_REGISTER];
```

```
//-------------------------------------------------
/**
  * Chế độ của thiết bị
  */
typedef enum {
  SETUP,
  RUN,
  SETUP_TO_RUN,
  RUN_TO_SETUP
} deviceStatus;
deviceStatus deviceMode = RUN;

//-------------------------------------------------
/**
  * Chế độ của setup
  * Bao gồm: Cấu hình id cho thiết bị, Cấu hình id cho biến tần
  */
typedef enum {
  SETUP_DEVICE_ID,
  SETUP_INVERTER_ID
} setupStatus;
setupStatus setupMode = SETUP_DEVICE_ID;

//-------------------------------------------------
/**
  * Biến toàn cục
  */
uint16_t lastDataFerq = 0, currentDataFreq = 0;
uint16_t lastDataControl = 0, currentDataControl = 0;
volatile int g_nIDDevice = 1;

//=================================================
// SETUP
//=================================================
void setup() {
  // Cấu hình bảng điều khiển trên board MCU
  pinMode(SETUP_PIN, INPUT_PULLUP);
  pinMode(PHA1_PIN, INPUT);
  pinMode(PHA2_PIN, INPUT);

  // Cấu hình Modbus Master
  MODBUS_MASTER_SERIAL.begin(MODBUS_MASTER_BAUD);
  mbm1.begin(MODBUS_INVERTER_SLAVE_ID, MODBUS_MASTER_SERIAL);

  // Cấu hình Modbus Slave
```

```
  mbs.configureHoldingRegisters(registerAddress, iLENGTH_REGISTER);
  MODBUS_SLAVE_SERIAL.begin(MODBUS_SLAVE_BAUD, MODBUS_SLAVE_CONFIG);
  mbs.begin(MODBUS_SLAVE_INIT_ID, MODBUS_SLAVE_BAUD);

  // Cấu hình LCD 16x2
  lcd.begin(16, 2);

  khoiDongDongCo();
}

//=================================================
// LOOP
//=================================================
void loop() {

  // Chế độ hệ thống
  switch (deviceMode) {
    case SETUP:
      {
        switch (setupMode) {
          case SETUP_DEVICE_ID:
            {
              // Hiển thị lên lcd
              static bool s_bEnableLcd = true;
              static int s_nLastIDDevice = g_nIDDevice;
              if (s_bEnableLcd) {
                lcd.clear();
                lcd.setCursor(0, 0);
                lcd.print("SET DEVICE ID:");
                lcd.setCursor(0, 1);
                lcd.print(g_nIDDevice);

                s_bEnableLcd = false;
              }
              if (s_nLastIDDevice != g_nIDDevice) {
                s_nLastIDDevice = g_nIDDevice;
                s_bEnableLcd = true;
              }

              // Đọc dữ liệu từ EEPROM
              static bool s_bEnableEEPROM = true;
              if (s_bEnableEEPROM) {
                g_nIDDevice = EEPROM.read(DEVICE_ID_ADDRESS_IN_EEPROM);

                s_bEnableEEPROM = false;
              }
```

```
    // Kiểm tra nút nhấn
    if (digitalRead(SETUP_PIN) == LOW) {
      delay(200);
      unsigned long l_nLasttime = millis();
      bool l_bChangeDeviceMode = false;

      // Nhấn giữ 3s
      while (digitalRead(SETUP_PIN) == LOW) {
        if (millis() - l_nLasttime >= 3000) {
          deviceMode = SETUP_TO_RUN;
          l_bChangeDeviceMode = true;
          break;
        }
      }

      if (l_bChangeDeviceMode == false) {
        // Cập nhật ID_Device vào EEPROM
        EEPROM.update(DEVICE_ID_ADDRESS_IN_EEPROM, g_nIDDevice);
        // Reset biến toàn cục ID
        s_bEnableEEPROM = true;
        s_bEnableLcd = true;
        setupMode = SETUP_INVERTER_ID;
        break;

      } else {
        // Cập nhật ID_Device vào EEPROM
        EEPROM.update(DEVICE_ID_ADDRESS_IN_EEPROM, g_nIDDevice);
        // Reset các biến cho phép hoạt động
        s_bEnableEEPROM = true;
        s_bEnableLcd = true;
        break;
      }
    }

    delay(10);

    break;
  }

case SETUP_INVERTER_ID:
  {
    // Hiển thị lên lcd
    static bool s_bEnableLcd = true;
    static int s_nLastIDDevice = g_nIDDevice;
```

```
if (s_bEnableLcd) {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("SET INVERTER ID:");
  lcd.setCursor(0, 1);
  lcd.print(g_nIDDevice);

  s_bEnableLcd = false;
}
if (s_nLastIDDevice != g_nIDDevice) {
  s_nLastIDDevice = g_nIDDevice;
  s_bEnableLcd = true;
}

// Đọc dữ liệu từ EEPROM
static bool s_bEnableEEPROM = true;
if (s_bEnableEEPROM) {
  g_nIDDevice = EEPROM.read(INVERTER_ID_ADDRESS_IN_EEPROM);

  s_bEnableEEPROM = false;
}

// Kiểm tra nút nhấn
if (digitalRead(SETUP_PIN) == LOW) {
  delay(200);
  unsigned long l_nLasttime = millis();
  bool l_bChangeDeviceMode = false;

  // Nhấn giữ 3s
  while (digitalRead(SETUP_PIN) == LOW) {
    if (millis() - l_nLasttime >= 3000) {
      deviceMode = SETUP_TO_RUN;
      l_bChangeDeviceMode = true;
      break;
    }
  }

  if (l_bChangeDeviceMode == false) {
    // Cập nhật ID_Device vào EEPROM
    EEPROM.update(INVERTER_ID_ADDRESS_IN_EEPROM, g_nIDDevice);
    // Reset các biến cho phép hoạt động
    s_bEnableEEPROM = true;
    s_bEnableLcd = true;
    setupMode = SETUP_DEVICE_ID;
    break;
```

```
            } else {
              // Cập nhật ID_Device vào EEPROM
              EEPROM.update(INVERTER_ID_ADDRESS_IN_EEPROM, g_nIDDevice);
              // Reset các biến cho phép hoạt động
              s_bEnableEEPROM = true;
              s_bEnableLcd = true;
              break;
            }
          }

          delay(10);

          break;
        }
      }

      break;
    }

  case SETUP_TO_RUN:
    {
      // Hủy ngắt ngoài
      detachInterrupt(0);

      // Cấu hình lại Modbus Master
      mbm1.begin(MODBUS_INVERTER_SLAVE_ID, MODBUS_MASTER_SERIAL);

      // Cấu hình lại Modbus Slave
      mbs.begin(MODBUS_SLAVE_INIT_ID, MODBUS_SLAVE_BAUD);

      deviceMode = RUN;

      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("----RUN--->");

      delay(3000);
      break;
    }


  case RUN:
    {
      // Kiểm tra nút nhấn
      if (digitalRead(SETUP_PIN) == LOW) {
        delay(200);
        unsigned long l_nLasttime = millis();
```

```
    // Nhấn giữ 3s
    while (digitalRead(SETUP_PIN) == LOW) {
      if (millis() - l_nLasttime >= 3000) {
        deviceMode = RUN_TO_SETUP;
        break;
      }
    }
  }

  // Chương trình thực thi
  mbs.poll();

  currentDataControl = registerAddress[iCONTROL_WORD];
  if (lastDataControl != currentDataControl) {
    switch (currentDataControl) {
      case QUAY_THUAN:
        quayThuan();
        break;

      case QUAY_NGHICH:
        quayNghich();
        break;

      case DUNG:
        dungDongCo();
        break;
    }

    lastDataControl = currentDataControl;
  }

  delay(100);

  currentDataFreq = registerAddress[iFREQUENCY_REFERENCE];
  if (lastDataFerq != currentDataFreq) {
    datTanSo(registerAddress[iFREQUENCY_REFERENCE]);
    lastDataFerq = currentDataFreq;
  }

  delay(100);

  if (mbm1.readHoldingRegisters(OUTPUT_FREQUENCY, 1) == 0) {
    registerAddress[iOUTPUT_FREQUENCY] = mbm1.getResponseBuffer(0);
    mbm1.clearResponseBuffer();

    if (registerAddress[iOUTPUT_FREQUENCY] > 200) {
      registerAddress[iOUTPUT_FREQUENCY] = 65536 - registerAddress[iOUTPUT_FREQUENCY];
```

```
        }

      }

      delay(100);

      if (mbm1.readHoldingRegisters(CURRENT_IN_THE_MOTOR, 1) == 0) {
        registerAddress[iCURRENT_IN_THE_MOTOR] = mbm1.getResponseBuffer(0);
        mbm1.clearResponseBuffer();
      }

      delay(100);

      if (mbm1.readHoldingRegisters(MOTOR_TORQUE, 1) == 0) {
        registerAddress[iMOTOR_TORQUE] = mbm1.getResponseBuffer(0);
        mbm1.clearResponseBuffer();
      }

      delay(100);

      // Hiển thị tần số lên LCD
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Tan so: ");
      lcd.print(String(registerAddress[iOUTPUT_FREQUENCY] * 0.1, 1));
      lcd.print(" Hz");
      lcd.setCursor(0, 1);
      lcd.print("Dong tai: ");
      lcd.print(String(registerAddress[iCURRENT_IN_THE_MOTOR] * 0.1, 1));
      lcd.print(" A");
      delay(5);

      break;
    }


  case RUN_TO_SETUP:
  {
    // Cho phép ngắt ngoài
    attachInterrupt(0, ISR_readVolumeEncoder, CHANGE);

    deviceMode = SETUP;

    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("----SETUP--->");

    delay(3000);
```

```
      break;
    }
  }
}
//=====================================================
// FUNTIONS
//=====================================================

//-----------------------------------------------------
// Điều khiển biến tần
void khoiDongDongCo(void) {
  mbm1.setTransmitBuffer(0, KHOI_DONG);
  mbm1.writeMultipleRegisters(CONTROL_WORD, 1);
  return;
}

void dungDongCo(void) {
  mbm1.setTransmitBuffer(0, DUNG);
  mbm1.writeMultipleRegisters(CONTROL_WORD, 1);
  return;
}

void quayThuan(void) {
  mbm1.setTransmitBuffer(0, QUAY_THUAN);
  mbm1.writeMultipleRegisters(CONTROL_WORD, 1);
  return;
}

void quayNghich(void) {
  mbm1.setTransmitBuffer(0, QUAY_NGHICH);
  mbm1.writeMultipleRegisters(CONTROL_WORD, 1);
  return;
}

void datTanSo(int l_nTanSo) {
  mbm1.setTransmitBuffer(0, 10 * l_nTanSo);
  mbm1.writeMultipleRegisters(FREQUENCY_REFERENCE, 1);
  return;
}

//-----------------------------------------------------
// Đọc Rotary Encoder
void ISR_readVolumeEncoder(void) {
  volatile static bool s_bEnable = true;
  volatile static bool s_bStartStatus = false, s_bCurrentStatus = false;
```

```
  if (s_bEnable) {
    s_bStartStatus = digitalRead(PHA1_PIN);
    s_bEnable = false;
  }

  s_bCurrentStatus = digitalRead(PHA1_PIN);
  if (s_bCurrentStatus != s_bStartStatus) {
    if (digitalRead(PHA2_PIN) != s_bCurrentStatus) g_nIDDevice++;
    else g_nIDDevice--;

    if (g_nIDDevice > 247) g_nIDDevice = 247;
    if (g_nIDDevice < 1) g_nIDDevice = 1;

    s_bEnable = true;
  }
}
```

2.3.2. Cấu hình OPC Server

Cấu hình giao thức truyền thông:

Identification	
Name	Channel1
Description	
Driver	Modbus RTU Serial
Diagnostics	
Diagnostics Capture	Disable

Connection Type	
Physical Medium	COM Port
Shared	No
Serial Port Settings	
COM ID	3
Baud Rate	9600
Data Bits	8
Parity	None
Stop Bits	1
Flow Control	None

a) **b)**

Hình 13 Cấu hình giao thức truyền thông

Chọn giao thức truyền thông Modbus RTU Serial như hình 13a. Sau đó cấu hình cổng giao tiếp như hình 13b: chọn cổng COM3, tốc độ baud là 9600, 8 bit data, không có bit parity và 1 bit stop.

Cấu hình cho thiết bị kết nối với OPC Server:

Identification	
Name	Device1
Description	
Driver	Modbus RTU Serial
Model	Modbus
Channel Assignment	Channel1
ID Format	Decimal
ID	1
Operating Mode	
Data Collection	Enable
Simulated	No

Hình 14 Cấu hình ID cho thiết bị kết nối

Trên hình 14, lựa chọn Channel của cài đặt là Channel1. Cài đặt ID cho thiết bị cần giao tiếp là 1 (có thể thay đổi sao cho đúng với giá trị được cài đặt trên board nhúng).

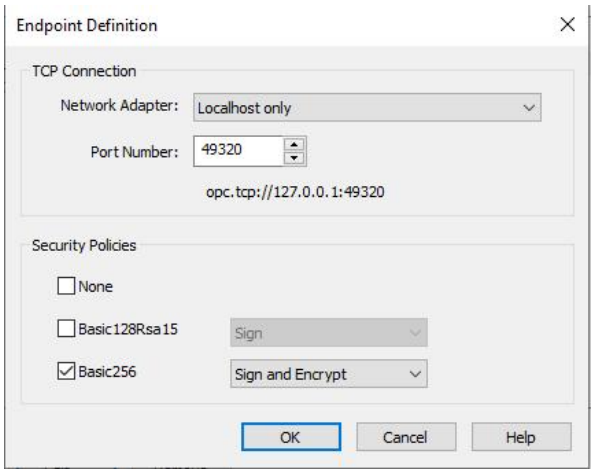
Tạo các Tags trên OPC Server:

Tag Name	Address	Data Type	Scan Rate
CONTROL_WORD	40001	Word	100
CURRENT_IN_THE_MOTOR	40004	Word	100
FREQUENCY_REFERENCE	40002	Word	100
MOTOR_TORQUE	40005	Word	100
OUTPUT_FREQUENCY	40003	Word	100

Hình 15 Các Tag trên OPC Server

Tạo các Tag trên OPC Server như hình 15, có địa chỉ tương ứng với các thanh ghi điều khiển đã được tạo trên borad nhúng, giúp truyền dữ liệu từ PC xuống vi điều khiển. Kiểu Tag có kiểu dữ liệu Word giúp dễ dàng đọc và ghi dữ liệu. Thời gian quét cho mỗi Tag là 100ms.

Cấu hình OPC Server để giao tiếp với Node-RED:



Hình 16 Địa chỉ OPC Server chạy trên máy tính

Trên hình 16, OPC Server chạy trên máy tính với cổng 49320 và có địa chỉ là ***opc.tcp://127.0.0.1:49320.*** Sử dụng phương thức bảo vệ là ***Basic:256***. Muốn kết nối với Server này, cần phải cấu hình trên Node-RED đúng địa chỉ và phương thức bảo vệ.

2.3.3. Chương trình điều khiển trên PC (Node-RED)

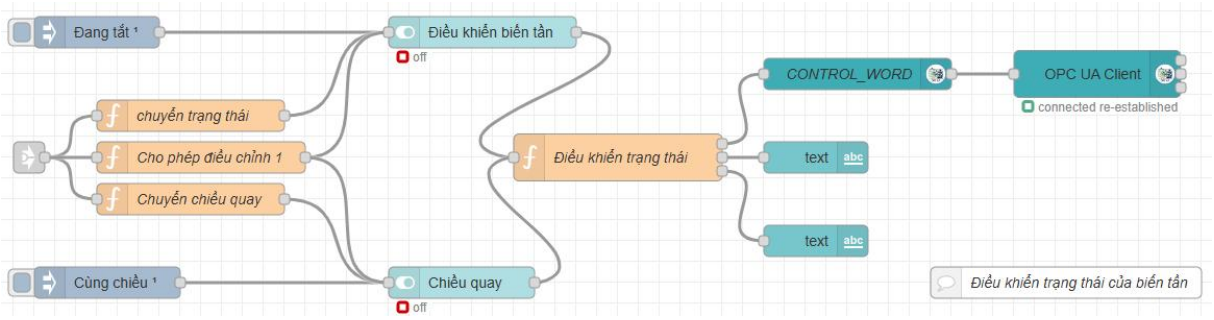
Mô tả chức năng chính:

- Cho phép người điều hành bật/tắt và thay đổi chiều quay của biến tần. Có thể điều chỉnh tần số thông qua thanh trượt Slider.
- Hiển thị các giá trị đọc được từ biến tần như: tần số thực tế, dòng tải và moment xoắn của động cơ.
- Cài đặt các khoảng thời gian hoạt động của biến tần, có thể tùy chỉnh bật/tắt, cài đặt chiều quay và tần số mong muốn. Khi đến khoảng thời gian được đặt trước,

bảng điều khiển thủ công sẽ bị vô hiệu hóa (có thể hủy chức năng này bất cứ khi nào người vận hành muốn), khi hết khoảng thời gian hoặc được hủy thủ công thì bảng điều khiển sẽ được mở trở lại.

Chương trình chính trên Node-RED:

- **Điều khiển trạng thái của biến tần:**



Hình 17 Chương trình điều khiển trạng thái trên Node-RED

Đoạn code trong node funtions ***điều khiển trạng thái:***

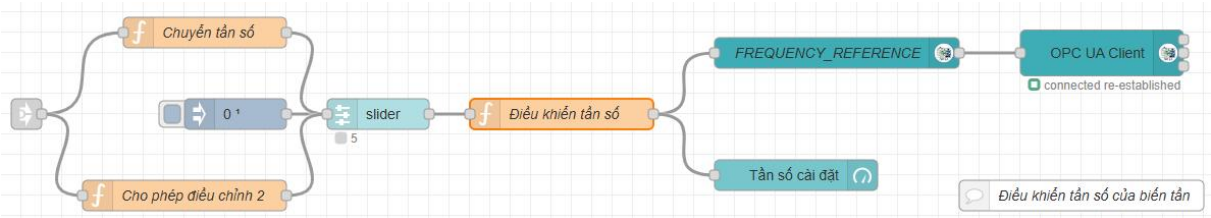
```
var str = msg.payload;

// Đọc giá trị của Switch điều khiển hoặc của chế độ hẹn giờ
// Để ghi giá trị vào biến "chieuQuay_cucBo"
if (str == "Ngược chiều") {
    global.set("chieuQuay_cucBo", str) || "Cùng chiều";
}
if (str == "Cùng chiều") {
    global.set("chieuQuay_cucBo", str) || "Cùng chiều";
}

// Đọc giá trị của Switch điều khiển hoặc của chế độ hẹn giờ
// Để ghi giá trị vào biến "trangThai_cucBo"
if (str == "Đang mở") {
    global.set("trangThai_cucBo", str) || "Đang tắt";
}
if (str == "Đang tắt") {
    global.set("trangThai_cucBo", str) || "Đang tắt";
}

// Ghi giá trị từ biến "chieuQuay_cucBo" và "trangThai_cucBo" vào OPC Server
if (global.get("trangThai_cucBo") == "Đang tắt") {
    msg.payload = 4111;
    var msg2 = { payload: "Đang tắt" };
    if (global.get("chieuQuay_cucBo") == "Cùng chiều") {
        var msg3 = { payload: "Cùng chiều" };
    }
    else {
        var msg3 = { payload: "Ngược chiều" };
    }
```

```
    return [msg, msg2, msg3];
}
else {
    if (global.get("chieuQuay_cucBo") == "Cùng chiều") {
        msg.payload = 15;
        var msg2 = { payload: "Đang mở" };
        var msg3 = { payload: "Cùng chiều" };
        return [msg, msg2, msg3];
    }
    else {
        msg.payload = 2063;
        var msg2 = { payload: "Đang mở" };
        var msg3 = { payload: "Ngược chiều" };;
        return [msg, msg2, msg3];
    }
}
```

Đoạn code trong node funtions ***chuyển trạng thái*** (các node chuyển đổi chiều quay và chuyển đổi tần số tương tự như node này):

```
// Nếu đến thời gian đã cài đặt trước
// Truyền trạng thái dã cài đặt vào biến "trangThai_cucBo"
if (global.get("enableKg1") == false) {
    msg.payload = global.get("trangThai1");
}
else if (global.get("enableKg2") == false) {
    msg.payload = global.get("trangThai2");
}
else if (global.get("enableKg3") == false) {
    msg.payload = global.get("trangThai3");
}
else {
    msg.payload = global.get("trangThai_cucBo"); // Ngược lại thì giữ nguyên
}

return msg;
```

- **Điều khiển tần số của biến tần:**



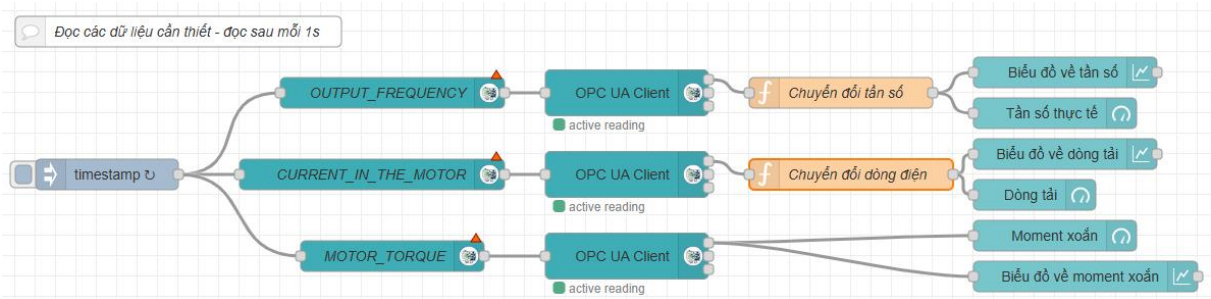
Hình 18 Chương trình điều khiển tần số trên Node-RED

Đoạn code trong node funtions ***điều khiển tần số***:

```
var num = msg.payload;
```

```
// Đọc giá trị của Slider điều khiển hoặc của chế độ hẹn giờ
// Để ghi giá trị vào biến "tanSo_cucBo"
global.set("tanSo_cucBo", num) || 0;
msg.payload = num
return msg;
```

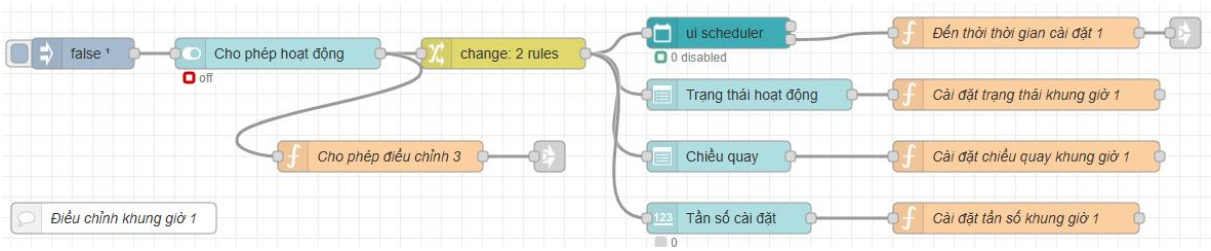
- **Đọc các giá trị thực tế từ biến tần:**



Hình 19 Chương trình đọc giá trị thực tế từ biến tần trên Node-RED

Sau mỗi giây, các giá trị sẽ được cập nhật.

- **Cài đặt khoảng thời gian điều khiển:**



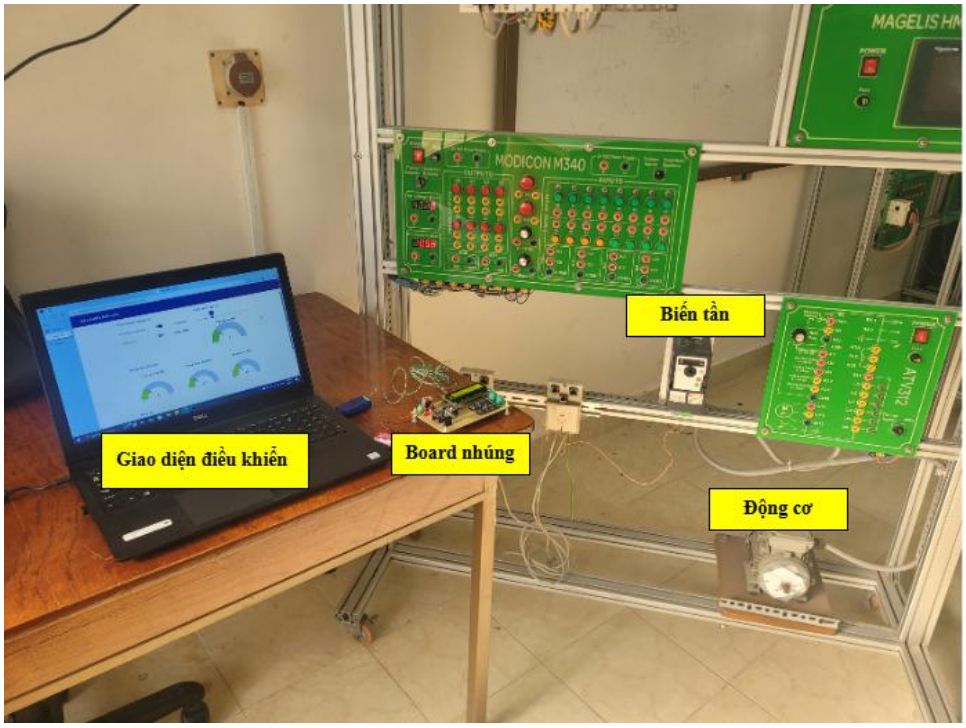
Hình 20 Chương trình cài đặt khoảng thời gian trên Node-RED

Đoạn code trong funtions ***đến thời gian cài đặt 1***:

```
var data = msg.payload;

// Đảo chiều để có thể bật tắt điều khiển thủ công
if (data == true) {
    data = false;
}
else {
    data = true;
}

// Lưu giá trị vào biến cho phép hoạt động của khung giờ 1
global.set("enableKg1", data) || true;

msg.payload = data;
return msg;
```

3. KẾT QUẢ THỰC HIỆN

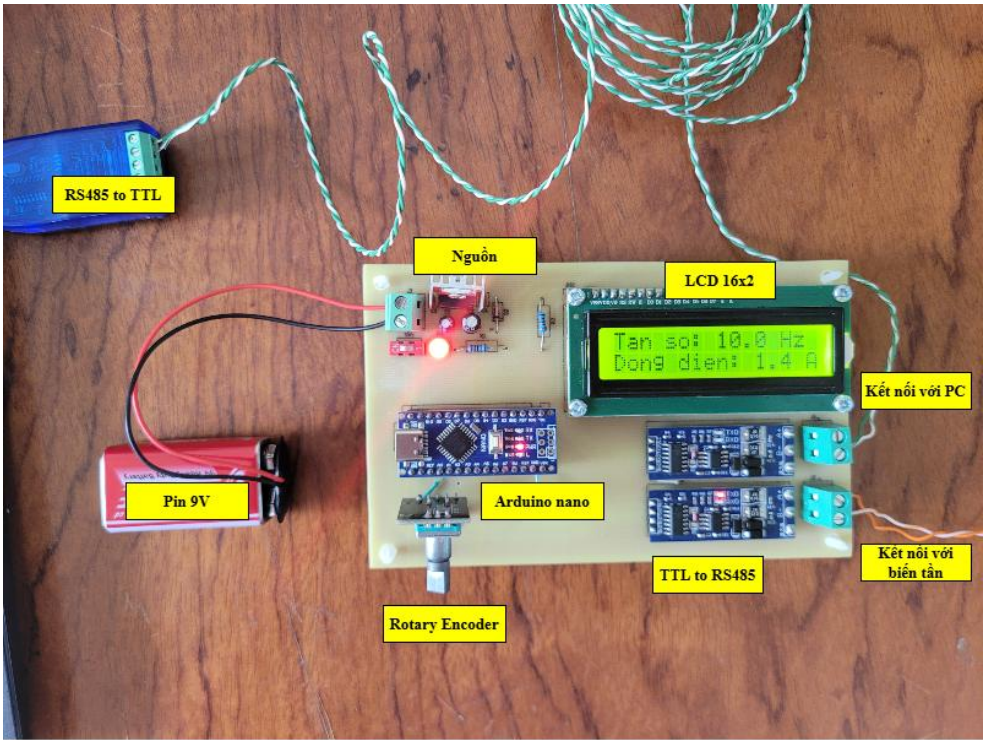
3.1. Mô hình thực tế



Hình 21 Mô hình thực tế

Mô hình thực tế được mô tả trong hình 21 bao gồm: máy tính điều khiển, board nhúng, biến tần ATV312 và động cơ điện 3 pha. Tất cả được kết nối bởi dây cáp xoắn đôi.

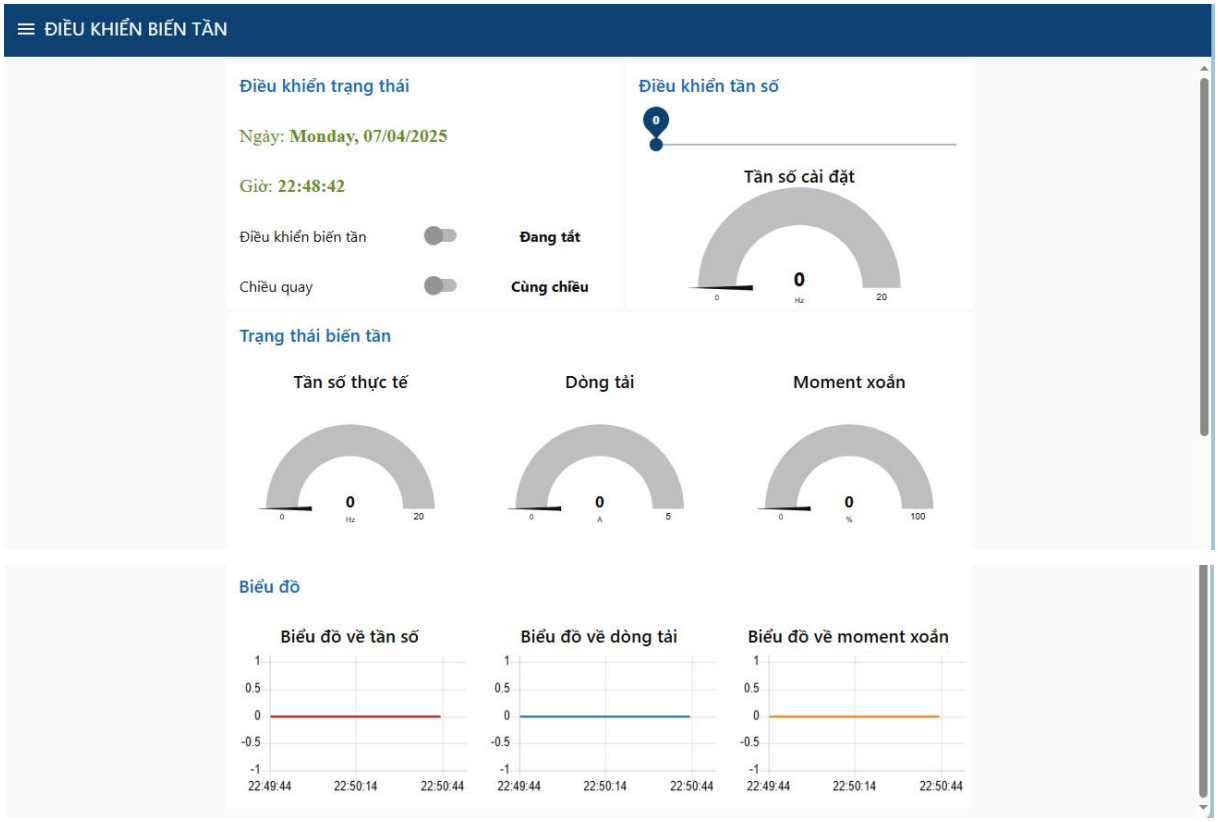
3.2. Phần cứng mạch điện



Hình 22 Mô hình phần cứng mạch điện

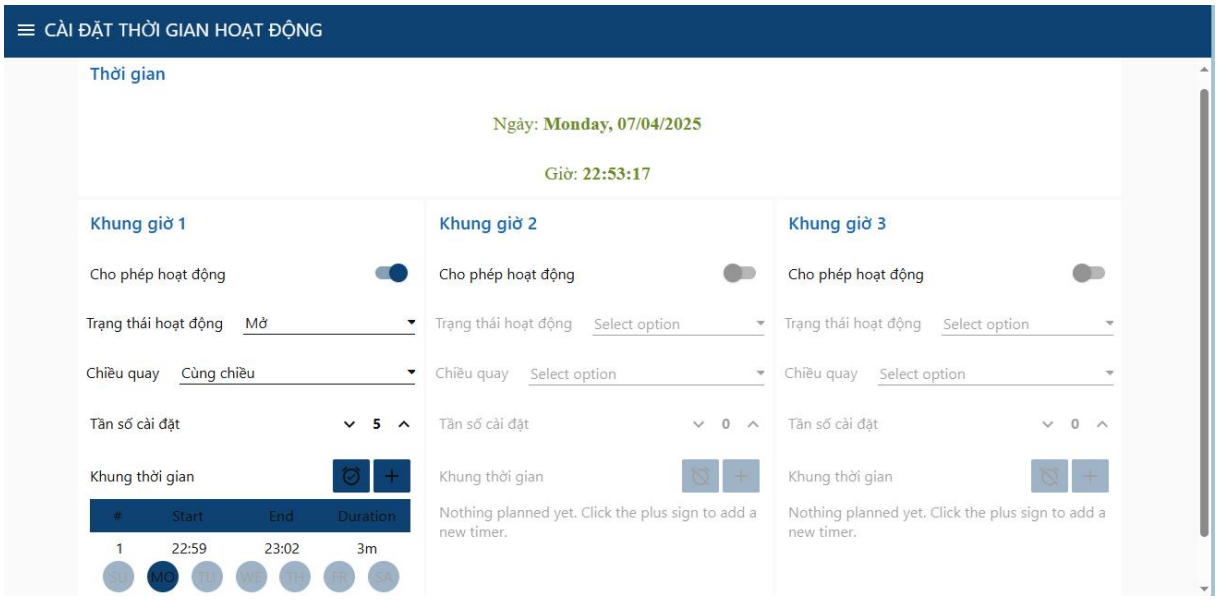
3.3. Giao diện điều khiển trên PC

Giao diện điều khiển thủ công:



Hình 23 Giao diện điều khiển biến tần

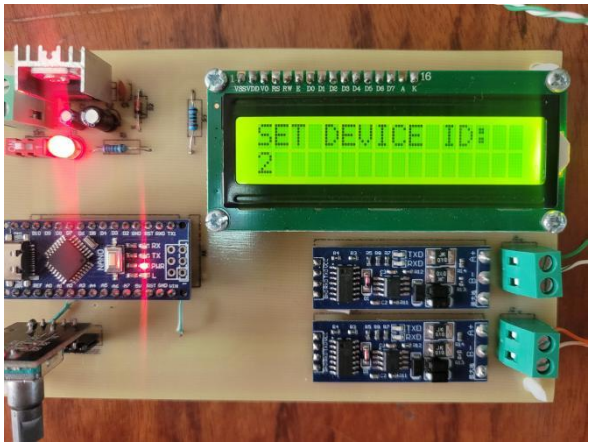
Giao diện cài đặt thời gian:

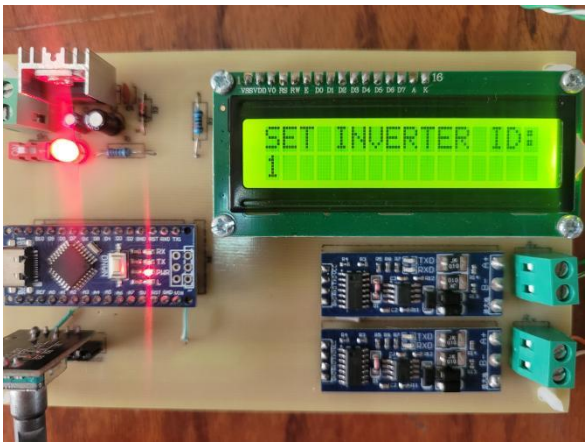


Hình 24 Giao diện cài đặt thời gian điều khiển biến tần

3.4. Kết quả hoạt động thực tế

Cài đặt ID trong giao thức truyền thông Modbus RTU





a) **b)**

Hình 25 Cài đặt ID cho thiết bị và biến tần

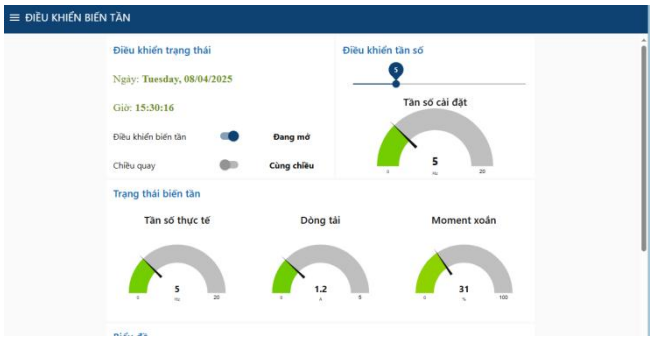
Trong hình 25a, cài đặt giá trị ID cho thiết bị là 2, như vậy để giao tiếp được với thiết bị này thông qua OPC Server, cần cấu hình ID thiết bị cần đọc trên Server là 2.

Cấu hình ID để giao tiếp với biến tần là 1 như hình 25b, do ID được cài đặt trên biến tần có giá trị là 1.

Sau khi cài đặt thành công, nhấn giữ phím trên module Rotary Encoder để bắt đầu khởi chạy thiết bị.

Bật/tắt biến tần

Bật biến tần:



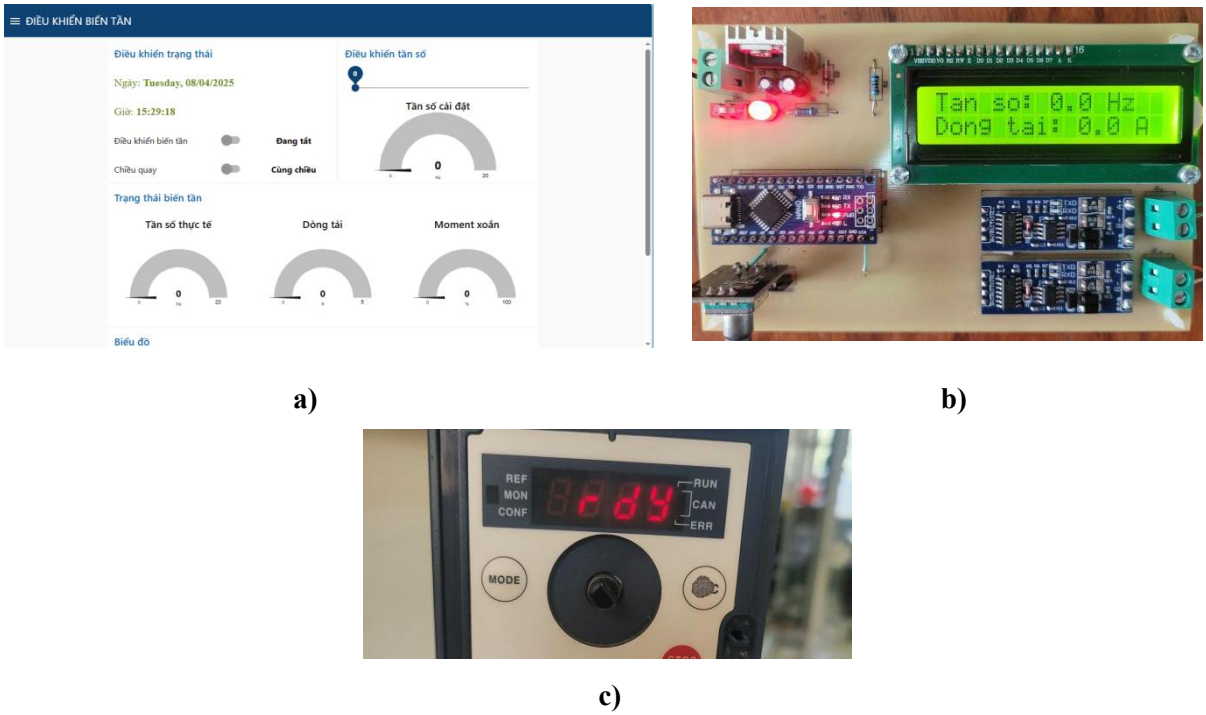
a) **b)**

c)

Hình 26 Thực hiện chức năng bật biến tần

- Điều khiển bật biến tần như hình 26a, đặt tần số là 5Hz và quay cùng chiều. Khi đó thiết bị điều khiển sẽ gửi lệnh cho biến tần hoạt động.
- Giá trị trên biến tần trong hình 26c là giá trị tần số thực tế của biến tần.

Tắt biến tần:



a) **b)**

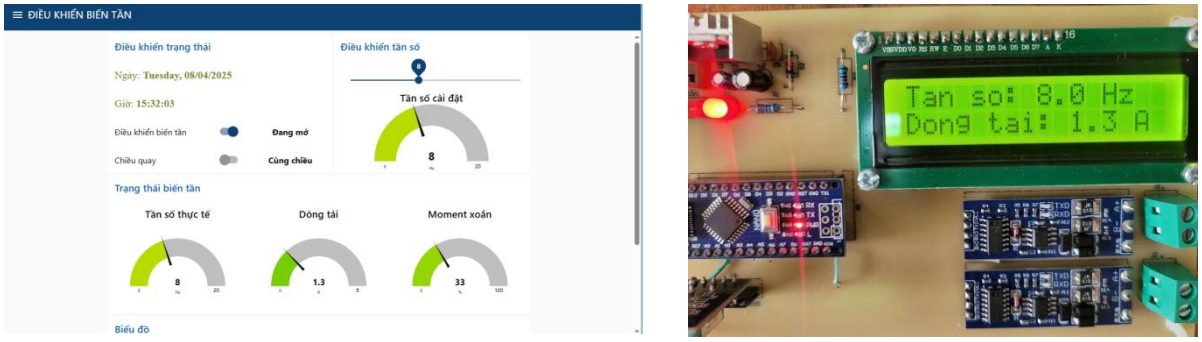
c)

Hình 27 Thực hiện chức năng tắt biến tần

- Điều khiển tắt biến tần như hình 27a, khi đó thiết bị điều khiển sẽ gửi lệnh dừng biến tần.
- Khi không hoạt động, biến tần sẽ hiện đoạn code như trên hình 27c.

Điều chỉnh chiều quay và tần số của biến tần

Quay cùng chiều - tần số 8Hz:



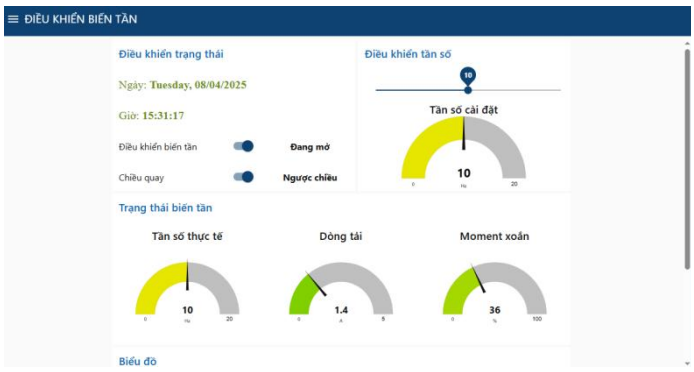
a) **b)**

c)

Hình 28 Thực hiện chức năng quay cùng chiều - 8Hz

- Cài đặt biến tần quay cùng chiều và đặt tần số 8Hz như hình 28a, khi đó biến tần sẽ nhận lệnh và đặt động cơ quay cùng chiều kim đồng hồ với tần số 8Hz.
- Khi quay cùng chiều, hiển thị tần số trên mặt đồng hồ là số dương như trên hình 28c.

Quay ngược chiều - tần số 10Hz:



a) **b)**

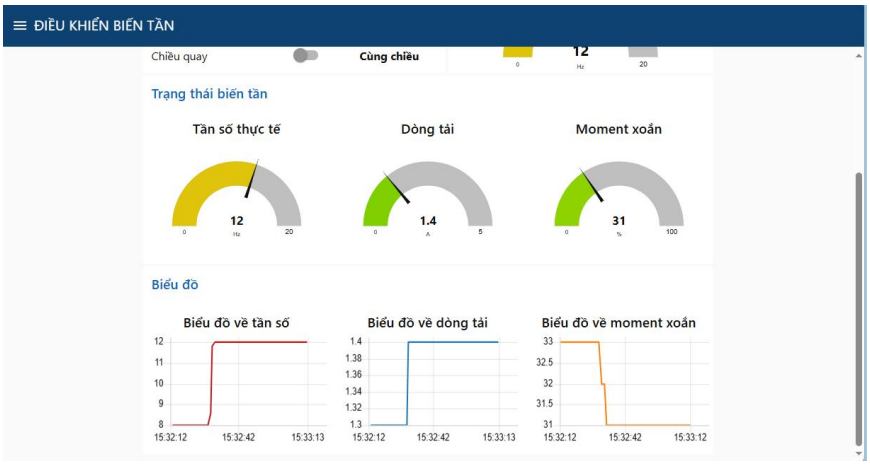
c)

Hình 29 Thực hiện chức năng quay ngược chiều - tần số 10Hz

Đặt biến tần quay ngược chiều với tần số 10Hz như trên hình 29a, thiết bị sẽ điều khiển động cơ quay ngược chiều kim đồng hồ.

Khi quay ngược chiều, giá trị tần số trên mặt đồng hồ của biến tần sẽ biểu diễn bằng số âm như hình 29c.

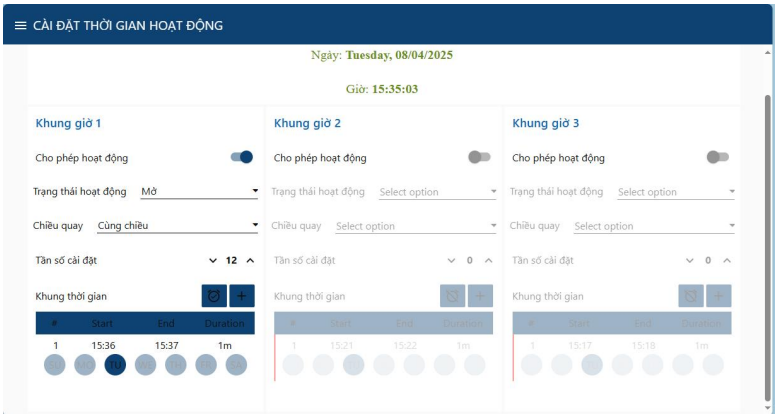
Đọc các giá trị gửi về hệ thống điều khiển



Hình 30 Thực hiện chứng năng đọc giá trị từ biến tần

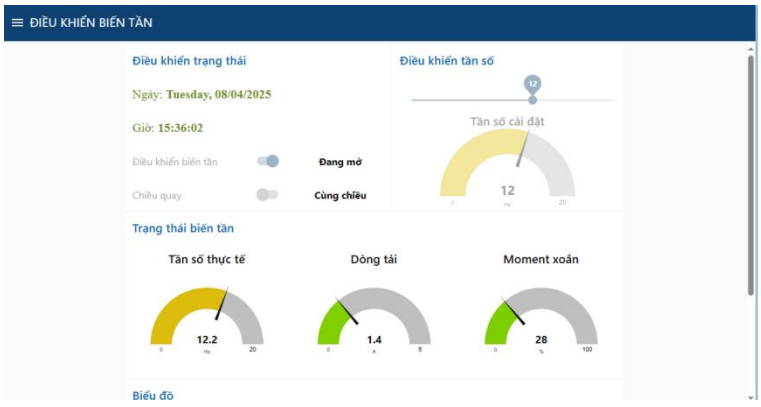
Cài đặt biến tần hoạt động cùng chiều với tần số 12Hz, giá trị phản hồi từ biến tần bao gồm: tần số thực tế là 12Hz, dòng tải là 1.4A và moment xoắn là 31% (dựa trên giá trị moment xoắn cực đại của động cơ).

Cài đặt các khung giờ hoạt động



Hình 31 Thực hiện chức năng đặt khung giờ hoạt động

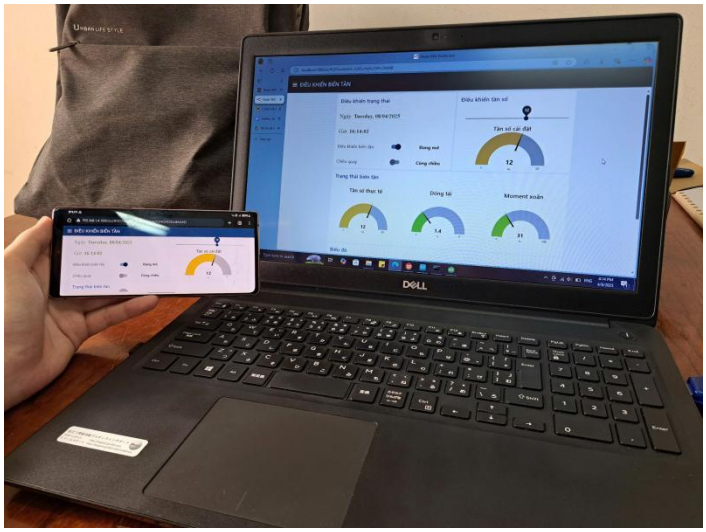
- Cho phép khung giờ 1 hoạt động như hình 31, cài đặt mở biến tần, quay cùng chiều và tần số là 12Hz. Thời gian hoạt động từ 15:36 đến 15:37 vào thứ 4 trong tuần.
- Thời điểm khi cài đặt khoảng thời gian này là thứ 4, 15:35.



Hình 32 Đến thời gian đã cài đặt trước

- Khi đến thời điểm được cài đặt trước (15:36), biến tần sẽ được điều khiển đúng như trạng thái và tần số đã được cài đặt trước đó. Đồng thời đóng bằng bảng điều khiển thủ công như hình 32.
- Đợi hết thời gian của khung giờ hoặc ngưng cho phép khung giờ 1 hoạt động nếu muốn sử dụng bảng điều khiển thủ công trở lại.

Điều khiển bởi nhiều thiết bị được kết nối với mạng cục bộ



Hình 33 Thực hiện chức năng truy cập bảng điều khiển bằng nhiều thiết bị

- Nhiều thiết bị được kết nối cùng một mạng Internet có thể truy cập vào bảng điều khiển được vận hành trên một máy chủ thông qua địa chỉ IP của máy chủ đó.
- Trên hình 33, sử dụng Laptop làm máy chủ và đồng thời có thể sử dụng điện thoại để truy cập vào bảng điều khiển của máy chủ đó.
- Có thể đặt mật khẩu, giới hạn số lượng truy cập vào máy chủ để bảm đảm an toàn cho hệ thống khi kết nối với mạng Internet không an toàn.

3.5. Ưu-nhược điểm của hệ thống

Ưu điểm:

- Điều khiển được các trạng thái cơ bản của biến tần, độ trễ thấp trong việc truyền nhận dữ liệu.
- Có thể cài đặt thời gian hoạt động cụ thể, giúp tự động hóa quá trình điều khiển.
- Có thể mở rộng điều khiển nhiều biến tần, đồng thời có thể điều khiển nhiều thiết bị khác mà thiết bị đó có sử dụng giao thức Modbus RTU.
- Có thể mở rộng thêm nhiều thiết bị điều khiển khác để kết nối với OPC Server, giúp tăng đáng kể cơ cấu chấp hành cần điều khiển.

Hạn chế:

- Chưa có tính năng cảnh báo về sự cố của biến tần.

- Chưa có kết nối với các thiết bị điều khiển hiện trường như HMI...
- Là mô hình thử nghiệm, chưa có các tính năng chống nhiễu mạnh mẽ, khó hoạt động ổn định khi vận hành trong môi trường thực tế.

3.6. Hướng phát triển

- Thêm các tính năng cảnh báo khi biến tần gặp sự cố.
- Bổ sung thêm tính năng Reset biến tần nếu gặp lỗi trong quá trình vận hành.
- Cải tiến thiết kế phần cứng, nhằm phù hợp hơn với môi trường làm việc nhà máy.

4. TÀI LIỆU THAM KHẢO

[1] ATV312 Modbus registers list. Schneider Electric.

[2] https://flowfuse.com/blog/2024/02/connect-node-red-to-kepware-opc/

[3] ThS.Nguyễn Khắc Nguyên. Bài giảng học phần Đo lường và điều khiển bằng máy tính, trường Đại học Cần Thơ.

5. PHỤ LỤC

5.1. Quá trình thực hiện đề tài

Kế hoạch thời gian

Bảng 1 Phân chia thời gian thực hiện đề tài

Thời gian	Nội dung công việc
17/1/2025	Nhận đề tài
18/1/2025 - 1/2/2025	Phân chia công việc tìm hiểu đề tài
2/2/2025 - 6/2/2025	Viết báo cáo kết quả tìm hiểu và hướng phát triển đề tài
7/2/2025	Báo cáo tiến độ thực hiện
8/2/2025 - 23/3/2025	Thực hiện đề tài với công cụ mô phỏng
24/3/2025 - 4/4/2025	Thực hiện đề tài với biến tần và động cơ thực tế
5/4/2025 - 9/4/2025	Viết báo cáo cuối kỳ
10/4/2025	Xem lại bài báo cáo, hiệu chỉnh mô hình nếu cần thiết
11/4/2025	Thực hiện báo cáo cuối kỳ

Thuận lợi và khó khăn trong quá trình thực hiện

Thuận lợi:

- Giảng viên hướng dẫn tận tình, định hướng rõ ràng trong quá trình thực hiện.
- Tài liệu liên quan đến biến tần, OPC Server và giao thức truyền thông Modbus RTU tương đối phong phú, dễ dàng tìm kiếm.
- Công cụ thực hiện như Node-RED, Arduino có cộng đồng hỗ trợ mạnh mẽ, giúp phát triển đề tài một cách nhanh chóng.

Khó khăn:

- Xây dựng mô hình trong thời gian ngắn, gây áp lực cho việc nghiên cứu lý thuyết và triển khai thực tế.
- Phòng thí nghiệm chỉ mở vào những khung giờ nhất định, do đó hạn chế phần nào khả năng xây dựng mô hình.

Kiến nghị:

Tăng cường thời gian để nhóm có thể nghiên cứu sâu hơn, và xây dựng hệ thống tối ưu hơn.

5.2. Danh sách các thành viên và bảng phân chia công việc

Bảng 2 Danh sách thành viên và bảng phân chia công việc

STT	Họ và tên	Nhiệm vụ	Đánh giá	Mức độ hoàn thành
1	Nguyễn Duy Tân	- Thiết kế board nhúng - Lập trình phần cứng - Lập trình cho giao diện điều khiển	Hoàn thành tốt	100%
2	Đặng Hoàng Khải	- Tìm hiểu cách thức truyền thông giữa OPC Server với PC và OPC Server với board nhúng - Lập trình giao diện điều khiển	Hoàn thành tốt	100%
3	Trương Duy Linh	- Tìm hiểu các chức năng trên Node-RED - Tạo giao diện điều khiển	Hoàn thành tốt	100%
4	Nguyễn Ngọc Tính	- Tìm hiểu về các thanh ghi trong biến tần ATV312 - Lập trình phần cứng	Hoàn thành tốt	100%

5.3. Trả lời câu hỏi

Nguyễn Duy Tân

Câu hỏi: Ưu - nhược điểm của cách thực hiện đề tài (đối với PC thì board điều khiển làm Slave, đối với biến tần thì board điều khiển làm Master)

Trả lời:

- Ưu điểm:
  + Tính linh hoạt cao: có thể lập trình xử lý các thuật toán điều khiển, phù hợp với yêu cầu của hệ thống mà không ảnh hưởng quá nhiều đến hệ thống chính. Có thể tích hợp với nhiều phần mềm giám sát hoặc SCADA một cách dễ dàng.
  + Khả năng mở rộng cao: Truyền thông Modbus trên chuẩn vật lý RS485 có số thiết bị kết nối tối đa là 32. Do đó để có thể kết nối nhiều thiết bị hiện trường với máy chủ, có thể sử dụng cách kết nối trên bài báo cáo.
- Nhược điểm:

Nếu vi điều khiển không có đủ tài nguyên xử lý, việc giao tiếp đồng thời giữa biến tần và PC có thể làm giảm hiệu suất của hệ thống.

Đặng Hoàng Khải

Câu hỏi: OPC sever kết nối với Slave qua các địa chỉ nào? Trong OPC sever sử dụng giao thức gì để kết nối và OPC sever đọc dữ liệu xuống sử dụng hàm function codes số mấy trong giao thức đó?

Trả lời:

- OPC sever kết nối với Slave qua các địa chỉ sau:
  + CONTROL_WORD với địa chỉ : 40001
  + CURRENT_IN_THE_MOTOR với địa chỉ: 40004
  + FREQUENCY_REFERENCE với địa chỉ: 40002
  + MOTOR_TORQUE với địa chỉ: 40005
  + UOTPUT_FREQUENCY với địa chỉ: 40003
- OPC sever sử dụng giao thức Modbus RTU Serial để kết nối với tốc độ baud là 9600, 8 bit data, không có bit parity và 1 bit stop.
- OPC sever đọc dữ liệu sử dụng hàm function codes 03h với tên hàm là Read Holding Registers, dải địa chỉ từ 40001 cho đến 49999 dùng để đọc các giá trị như : tốc độ động cơ, tần số, điện áp, dòng điện, trạng thái động cơ, giá trị cài đặt đang chạy,....

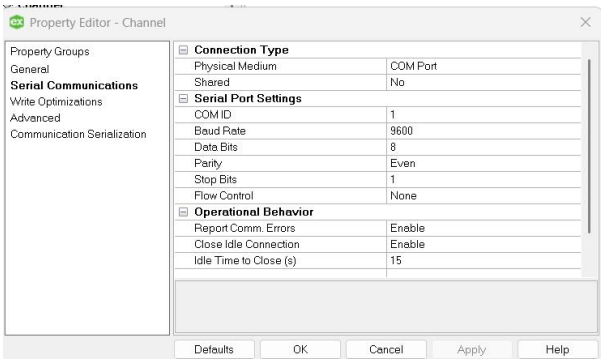
Trương Duy Linh

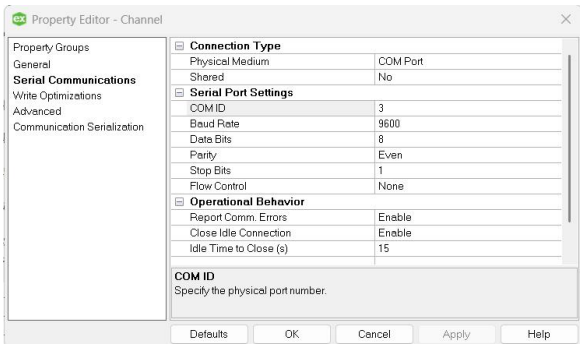
Câu hỏi: Cấu hình OPC Sever như thế nào để quản lí nhiều Board với mỗi Board có thể điều khiển nhiều biến tần (VD: Điều khiển 5 Board, mỗi Board điều khiển 10 biến tần) ?

Trả lời: Ta thực hiện cấu hình OPC Server theo 4 bước sau:

Bước 1: Cấu hình giao thức truyền thông (Tạo 1 channel):

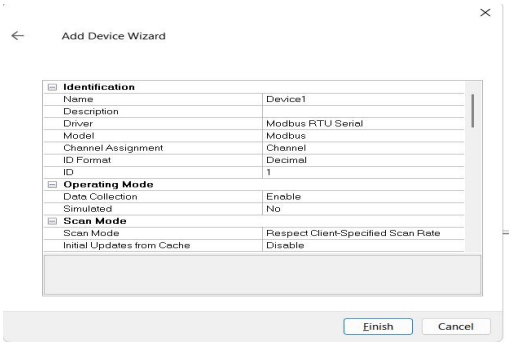
- Type: Modbus RTU Serial.
- COM Port: chọn đúng COM (COM3).
- Baud Rate: giống Arduino (9600).
- Parity: None (phải giống Arduino).
- Stop Bits: 1.
- Data Bits: 8.
- RTU Mode: Yes (được bật sẵn trong Kepware).





Bước 2: Cấu hình cho thiết bị kết nối với OPC Server (Board Arduino) :

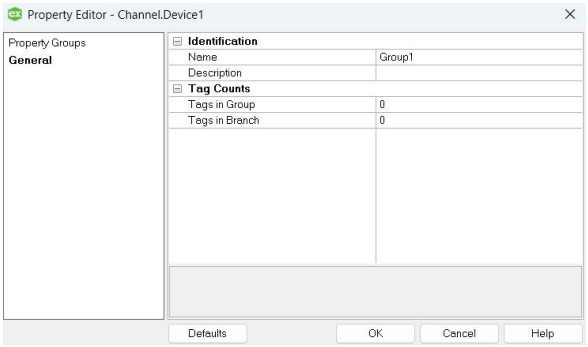
- Tạo 5 Device:
  + Device Name: Board_1, Device ID: 1.
  + Device Name: Board_2, Device ID: 2.
  + Device Name: Board_3, Device ID: 3.
  + Device Name: Board_4, Device ID: 4.
  + Device Name: Board_5, Device ID: 5.
- Giữ nguyên cấu hình còn lại theo mặc định có thể điều chỉnh Scan Rate để thay đổi tốc độ đọc dữ liệu.



- Cấu hình tương tự cho các Device còn lại, chỉ đổi Device ID : 1-5.

Bước 3: Cấu hình các Tag cho từng Device :

- Mỗi Arduino điều khiển 10 biến tần, mỗi biến tần dùng 5 thanh ghi để đọc và ghi dữ liệu:
  + Tạo 10 Group Tags (đặt tên là Group 1-10).

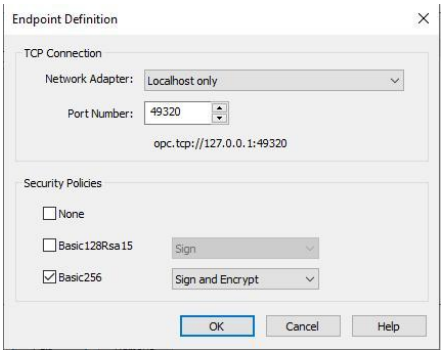


  + Trong mỗi Group Tags ta tạo 5 Tag như sau (các Tags này có địa chỉ tương ứng với các thanh ghi điều khiển đã được tạo trên borad nhúng):

Tag Name	Address	Data Type	Scan Rate
CONTROL_WORD	40001	Word	100
CURRENT_IN_THE_MOTOR	40004	Word	100
FREQUENCY_REFERENCE	40002	Word	100
MOTOR_TORQUE	40005	Word	100
OUTPUT_FREQUENCY	40003	Word	100

  + Các Group Tags còn lại cũng tạo 5 Tags tương tự nhưng Address (40001-40050).
- Trên đây là các bước để cấu hình cho Device1, các Device còn lại cũng cấu hình tương tự (Do khác device nên không cần đổi Address cho các Tags).

Bước 4: Cấu hình OPC Server để giao tiếp với Node-RED:



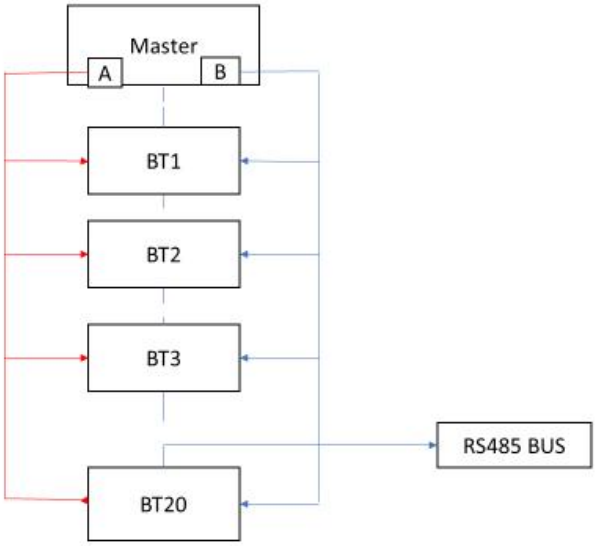
OPC Server chạy trên máy tính với cổng 49320 và có địa chỉ là opc.tcp://127.0.0.1:49320. Sử dụng phương thức bảo vệ là Basic:256. Muốn kết nối với Server này, cần phải cấu hình trên Node-RED đúng địa chỉ và phương thức bảo vệ.

Nguyễn Ngọc Tính

Câu hỏi: Sơ đồ phần cứng Master đấu nối với 20 biến tần và cấu hình biến tần với Master.

Trả lời:

- Sơ đồ phần cứng:



- Cấu hình:
  + Cấu hình 20 biến tần:

Biến tần	Địa chỉ Modbus	Baudrate	Parity	Stop bit
1	1	9600	None	1
2	2	9600	None	1
3	3	9600	None	1
…….	……..	…..	……	…….
20	20	9600	None	1

  + Cấu hình Master:

Thông số	Giá trị
Giao thức	Modbus RTU
Cổng truyền	RS485
Baudrate	9600
9600	9600
Data bit	8
Stop bit	1