ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

ĐỒ ÁN MÔN HỌC

Cân điện tử

Nguyễn Văn Minh – 20215092

Ngành CNTT Việt Nhật (HEDSPI)

Giảng viên hướng dẫn: Nguyễn Đức Tiến

Bộ môn: Kỹ thuật máy tính

Hà Nội, tháng 5 năm 2024

MỤC LỤC

CHƯƠNG 1: Giới thiệu sản phẩm

Tính năng sản phẩm

- Cân khối lượng: hiển thị kết quả tới 2 chữ số thập phân, cho phép hiển thị kết quả âm
- Tinh chỉnh kết quả nếu cân chưa đúng
- Thay đổi đơn vị cân từ kilogram sang pound
- Điều chỉnh cân về 0 kg
- Xem kết quả của 3 lượt cân gần nhất
- Tự động tắt đèn nền màn hình khi không sử dụng

Thiết bị sử dụng

- Board: ESP32-C3, Wemos-S2-mini
- 1 mạch khuyếch đại HX711 + 4 load-cell 50kg
- LCD 16x2 I2C hoặc SSD1306 OLED
- 5 công tắc:

  + UP, DOWN: Dùng để tinh chỉnh kết quả nếu cân hiển thị khối lượng chưa đúng

  + MODE: Thay đổi đơn vị cân từ kilogram sang pound

  + TAREE: Điều chỉnh cân về 0 kg

  + DATA: Xem kết quả lần lượt của 3 lần cân gần nhất

Môi trường phát triển

- Visual Studio Code, Arduino IDE: biên soạn mã nguồn
- Wokwi: chạy mô phỏng sản phẩm

Kỳ vọng phát triển sản phẩm

- Bluetooth để gửi dữ liệu ra thiết bị khác
- Bổ sung thêm nút bấm điều khiển.
- Kết nối nối máy đọc thẻ NFC, RFID
- Kết hợp ESP32-Cam để chụp hình người cân, nhận dạng ảnh
- Kết nối WiFi, gửi dữ liệu về datacenter
- Biến thành thiết bị điều khiển joystick thông qua HID
- Cảm biến đo mạch máu và nông độ oxy

CHƯƠNG 2: Xây dựng chương trình

Giao tiếp với HX711

Chương trình xử dụng thư viện HX711-HEDSPI.h, thư viện này định nghĩa lớp HX711 như sau:

```
#define GAIN_128 25
#define GAIN_64 27
#define GAIN_32 26
#define getData() getData_H(Gain)

class HX711 {
public:
  HX711(byte DOUT, byte PD_SCK, byte Gain = GAIN_128, float Scale = 1.0f);
  void init();

  void setGain(byte Gain);
  void setScale(float Scale);
  void setZero(int32_t Zero);
  int32_t setZero();
  int32_t getData_H(byte Gain = GAIN_128, uint16_t check_freq = 100);
  int32_t getData_L(byte Gain = GAIN_128, uint16_t check_freq = 100);
  float getWeight();

private:
  byte DOUT;
  byte PD_SCK;
  byte Gain = GAIN_128;
  int32_t Zero = 0;
  float Scale = 1;
};
```

- Giải thích:

+ DOUT, PD_SCK: 2 chân tín hiệu nối với Mạch chuyển đổi ADC 24 bit HX711

+ Gain: hệ số khuyếch đại của HX711

+ Zero: Zero được thiết lập bằng giá trị cân hiện tại. Để điều chỉnh hiển thị khối lượng về 0, chỉ cần trả về khối lượng đo được trừ đi Zero.

+ Scale: HX711 trả về một giá trị tỷ lệ thuận với khối lượng đo được. Scale là giá trị đó

+ Hàm getData_H(): Thực hiện gửi tín hiệu điều khiển ra chân PD_SCK và đọc giá trị khuyếch đại thu được từ chân DOUT. Thứ tự gửi nhận tín hiệu như sau

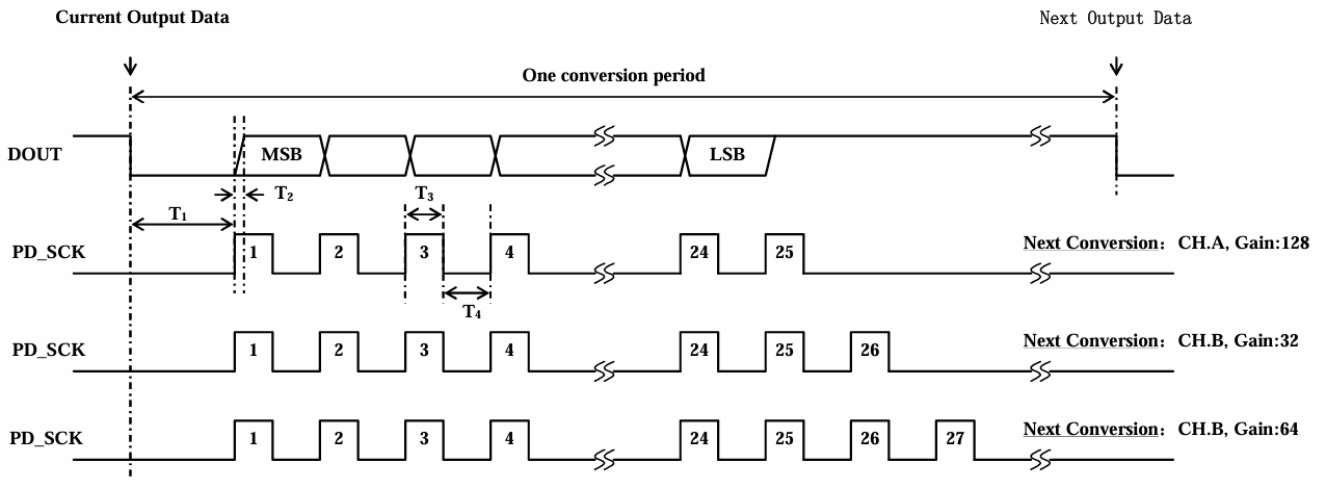

Fig.1 Data output, input and gain selection timing and control

1. Chờ tới khi (DOUT → LOW) thì bắt đầu đọc dữ liệu
2. Đọc 24 bit giá trị khuyếch đại. Mỗi khi phát một xung LOW → HIGH thì đọc 1 bit từ chân DOUT
3. Phát thêm (Gain – 24) xung để set up giá trị khuyếch đại của HX711 cho lần đọc tiếp theo

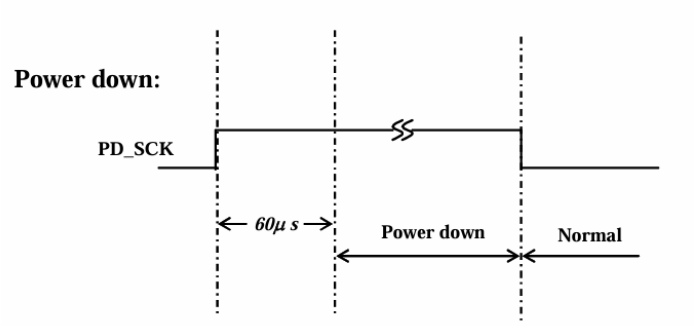

Fig.2 Power down control

4. (Option) Đưa PD_SCK lên HIGH trong tối thiểu 60µs để đưa HX711 vào chế độ Power down

Sau quá trình đọc dữ liệu, nếu DOUT không lên mức HIGH thì trả về 0x7fffff như một giá trị để thông báo lỗi. Mặc khác, nếu DOUT luôn bằng HIGH trong quá trình đọc thì hàm trả về -1 là giá trị đọc bị lỗi.

Qua quá trình kiểm thử việc đọc dữ liệu ta thu được kết quả sau:

Gain	Giá trị của PD_SCK sau khi đọc	Trung bình thời gian mỗi lần đọc
128	HIGH	99ms
32	HIGH	49ms
128	LOW	100ms
32	LOW	100ms

Tổng quan về chương trình

Sau đây là mô tả về chương trình được xây dựng cho ESP32-C3 và LCD1602:

Lấy dữ liệu

- Sử dụng hàm getData_Avg() để lấy dữ liệu từ HX711. Các bước thực hiện như sau:
    1. Hàm này sử dụng mảng d[N] lấy N dữ liệu (không lấy -1 và các giá trị quy đổi ra khối lượng lớn hơn tải trọng tối đa MAX_LOAD, như thế đồng thời sẽ không lấy giá trị 0x7fffff).
    2. Lấy giá trị trung bình d_avg của N phần tử trong mảng d[]. Sắp xếp mảng d[] theo thứ tự các phần tử cách xa dần giá trị trung bình d_avg.
    3. Gán d_worst = d[N-1] là phần tử có giá trị cách xa d_avg nhất.
    4. Thực hiện tối đa K lần dữ liệu tới khi d_worst < sai số tuyệt đối Absolute_error. Nếu dữ liệu lấy được gần d_avg hơn d_worst hiện tại thì thay thế phần tử có khoảng cách d_worst trong mảng bằng phần tử vừa lấy, đồng thời sắp xếp lại mảng và cập nhật lại d_avg, d_worst.
    5. Cập nhật sai số sensor_error = d_worst và trả về d_avg.

- Hàm getData_Avg() có thể trả về giá trị lỗi 0x7fffff hoặc giá trị có sai số lớn (sensor_error > Absolute_error) nên cần hàm getData_() để lấy lại số liệu. Hàm getData_() được xây dựng với 1 tham số đầu vào là allow_delay. Nếu allow_delay == 0 thì gọi lại hàm getData_Avg() để lấy lại dữ liệu một lần nữa. Nếu allow_delay == 1 thì thực hiện lấy lại dữ liệu tối đa 2 lần, mỗi lần cách nhau 105ms. Nếu sau các bước lấy lại dữ liệu mà dữ liệu nhận được lỗi (0x7fffff) thì trả về giá trị lấy thành công trước đó và in ra thông báo lỗi qua Serial.

Hiển thị kết quả

Hiện thị kết quả dưới dạng số thập phân 2 chữ số sau dấu phẩy, cho phép hiển thị số âm. Có 2 lựa chọn hiển thị đơn vị là kilogram-kg hoặc pound-lb. Giá trị hiển thị bằng dữ liệu lấy được từ bộ khuyếch đại trừ cho giá trị dữ liệu tương ứng với 0kg, sau đó chia cho hệ số giữa dữ liệu thu được từ bộ khuyếch đại và khối lượng thực tế.

data: Giá trị đọc được từ HX711

m: Khối lượng thực tế

w: Giá trị hiển thị

Zero: Giá trị data tương ứng với w = 0 (m có thể khác 0kg)

$\Longrightarrow$ Scale = data / m

$\Longrightarrow$ w = (data – Zero) / Scale

Với chương trình lập trình cho LCD1602, dòng đầu tiên hiển thị tiêu đề MAIN_TITLE của chương trình, dòng thứ 2 hiển thị kết quả cân. Dưới đây là hàm dùng để hiển thị kết quả cân lên màn hình.

Xử lý ngắt

Hàm xử lý ngắt cho mỗi nút bấm có cấu trúc tương tự nhau. Để tránh hiện tượng tín hiệu không ổn định trong thời gian bấm nút, chương trình chỉ ghi nhận mỗi ngắt cách nhau một khoảng thời gian cố định DEBOUNCE_TIME (200ms). Khi ghi nhận ngắt của nút nào thì gán biến tương ứng bằng 1, tăng biến ngắt tổng của chương trình (interrupt) lên 1 đơn vị.

```
void IRAM_ATTR modeISR()
{
 if (millis() - prev_press < DEBOUNCE_TIME)
   return;
 prev_press = millis();
 mode = 1;
 interrupt++;
}
```

Sau khi ghi nhận ngắt, chương trình chính trong hàm loop() sẽ kiểm tra các ngắt, xử lý và lưu lại những ngắt nào được xử lý qua bộ các biến _up, _down, ...

```
while (tare == 1 || mode == 1 || up == 1 || down == 1 || record == 1)
 { // Continue Handling
```

- Xử lý ngắt cho nút UP (tương tự cho nút DOWN): nút UP được dùng để điều chỉnh kết quả hiện thị tăng lên nếu kết quả hiện thị đang bị lệch bằng cách giảm hệ số Scale

xuống. Nếu người dùng nhấn nút 1 lần, khối lượng hiển thị sẽ chỉ tăng một ít. Nếu người dùng bấm giữ nút UP thì khối lượng hiển thị sẽ tăng nhanh hơn.

- Xử lý ngắt cho nút TARE: đọc dữ liệu từ bộ khuyếch đại và gán cho biến Zero.
- Xử lý ngắt cho nút MODE: thay đổi biến Mode từ KG_MODE thành LB_MODE hoặc ngược lại.
- Xử lý ngắt cho nút RECORD: hiển thị lần lượt khối lượng cân được của 3 lần gần nhất, mỗi lần bấm nút chỉ hiển thị một giá trị, nếu bấm nút lần thứ 4 thì quay trở lại chương trình chính.

Chế độ ngủ

Nếu sau một khoảng thời gian cố định, giá trị đọc được từ bộ khuyếch đại không thay đổi thì chương trình sẽ điều khiển màn hình nhấp nháy 2 lần, sau đó để hiển thị thêm trong một khoảng thời gian rồi tắt màn hình.
Trong lúc chương trình đang chuẩn bị vào chế độ ngủ, chương trình vẫn lắng nghe có ngắt xảy ra hay không, khối lượng có thay đổi hay không. Nếu phát hiện có ngắt hoặc thay đổi khối lượng, chương trình sẽ trở về luồng hoạt động chính. Thực hiện điều này bằng cách sử dụng hàm delay_W() tự xây dựng thay vì dùng hàm delay() thông thường. Ngoài ra khi vào chế độ ngủ, chương trình sẽ chuyển sang dùng GAIN_64 thay vì GAIN_128 để giảm thời gian đọc dữ liệu từ bộ khuyếch đại.

Đánh giá sản phẩm

Các vấn đề gặp phải

1. Đọc dữ liệu từ bộ khuyếch đại HX711 mất khá nhiều thời gian (với GAIN_128 thì trung bình mất 100ms cho mỗi lần đọc). Mà chương trình cần đọc nhiều lần liên tiếp để đảm bảo lấy được giá trị có độ chính xác cao. Vì thế khi chương trình vừa cần delay để hiển thị kết quả, vừa cần lắng nghe xem khối lượng có thay đổi không để hiển thị lại ngay thì thời gian mỗi lần delay chưa được chính xác.
2. Không thể đưa toàn bộ phần xử ngắt vào hàm xử lý ngắt vì trong hàm xử lý ngắt không thực hiện được hàm delay() và các hàm giao tiếp với màn hình lcd, oled.
→Theo như em tìm hiểu thì có thể là do trong khi thực hiện một hàm ngắt, chương trình tạm thời tắt tất cả các ngắt khác của hệ thống. Đồng thời có thể có sự tối ưu code trong quá trình biên dịch dẫn đến sai logic của người lập trình.

Hướng phát triển

- Cải thiện tính dễ sử dụng: có thể thay nút bấm bằng các nút cảm ứng ở mặt trên của cân hoặc điều khiển bằng điện thoại.
- Thêm tính năng: phát triển tính năng nhận diện (bằng camera nhận diện khuôn mặt hoặc nhận diện qua ứng dụng trên điện thoại) để lưu kết quả cân của từng người; đánh giá chỉ số BMI; vẽ biểu đồ thể hiện sự thay đổi cân nặng qua các lần đo.
- Giảm thời gian phản hồi: phát triển thêm thuật toán đánh giá mức độ ổn định, sai số của cảm biến và bộ khuyếch đại để điều chỉnh tần suất giao tiếp với bộ khuyếch đại.
- Cải thiện giao tiếp người dùng: thêm tính năng thay đổi kiểu hiện thị như thay đổi phông chữ hay thay đổi tiêu đề hiển thị.

Phụ lục

Mã nguồn sản phẩm

Cung cấp mã nguồn cho các phiên bản phần cứng sau:

1. ESP32-C3, LCD1602
2. ESP32-C3, OLED SSD1306
3. Wemos S2 mini, LCD1602
4. Wemos S2 mini, OLED SSD1306

Download mã nguồn tại: VanMinh153/HX711_Digital_Scale (github.com)

Hướng dẫn chạy mô phỏng sản phẩm

Môi trường cần thiết:

+ Visual Studio Code có cài extension Arduino, Wokwi
+ Các thư viện hệ thống của ESP32

Các bước thực hiện:

1. Copy nội dung file mã nguồn và file diagram của phiên bản tương ứng trong thư mục release vào file main.ino và file diagram.json
(Lưu ý không thay đổi tên file main.ino và diagram.json)
2. Mở file main.ino, bấm F1, chọn Arduino: Verify và đợi quá trình biên dịch hoàn tất
3. Bấm F1, chọn Wokwi: Start Simulator

Các hàm sử dụng trong chương trình

HX711::getData_H()

```
/**
 * @brief     Get data from HX711 and set up PD_SCK = HIGH after
 */
int32_t HX711::getData_H(byte gain, uint16_t check_freq)
{
  const byte response_time = 1;
  digitalWrite(PD_SCK, LOW);

  unsigned long timer = millis();
  while (digitalRead(DOUT) == HIGH && millis() - timer < 102)
    delayMicroseconds(check_freq);

  int32_t data = 0;
  for (uint8_t i = 0; i < 24; i++)
  {
```

```
  digitalWrite(PD_SCK, HIGH);
  delayMicroseconds(response_time);
  data = ((data << 1) | digitalRead(DOUT));
  digitalWrite(PD_SCK, LOW);
  delayMicroseconds(response_time);
 }

 digitalWrite(PD_SCK, HIGH);
 delayMicroseconds(response_time);
 if (digitalRead(DOUT) == LOW)
  return 0x7fffff;

 byte i = gain - 25;
 while (i > 0)
 {
  i--;
  digitalWrite(PD_SCK, LOW);
  delayMicroseconds(response_time);
  digitalWrite(PD_SCK, HIGH);
  delayMicroseconds(response_time);
 }
 delayMicroseconds(65);

 if (bitRead(data, 23) == 1)
  data |= 0xFF000000;

 return data;
}
```

getData_Avg()

```
int32_t getData_Avg()
{
 const byte N = 5;
 const byte K = 5;
 int32_t d[N];
 int32_t d_avg = 0;
 uint32_t d_worst = 0;
 int32_t d_temp = 0;
 byte count = 0;
 byte countZ = 0;

 for (byte i = 0; i < 2 * N; i++)
 {
  d_temp = sensor.getData();
  if (d_temp == -1)
   continue;
  if (d_temp == prev_getData)
```

```
    {
      countZ++;
      if (countZ == 2 && count == 0)
        return d_temp;
      continue;
    }
    if (abs(d_temp) < Scale * MAX_LOAD)
    {
      d[count] = d_temp;
      d_avg += d_temp;
      count++;
      if (count == N)
        break;
    }
  }

  if (count < N)
    return 0x7fffff;
  d_avg /= N;
  sort_(d, N, d_avg);
  d_worst = abs(d[N - 1] - d_avg);

  count = 0;
  while (count < K && d_worst > Absolute_error)
  {
    d_temp = sensor.getData();
    if (d_temp == -1)
      continue;
    if (abs(d_temp - d_avg) < d_worst)
    {
      d_avg += (d_temp - d[N - 1]) / N;
      d[N - 1] = d_temp;
      sort_(d, N, d_avg);
      d_worst = abs(d[N - 1] - d_avg);
    }
    count++;
  }

  sensor_error = d_worst;
  prev_getData = d_avg;
  return d_avg;
}
```

getData_()

```
int32_t getData_(byte allow_delay)
{
```

```
 int32_t d = getData_Avg();
 if (d != 0x7fffff && sensor_error < Absolute_error)
   return d;

 if (allow_delay == 0)
   d = getData_Avg();

 if (allow_delay == 1)
 {
   for (byte i = 0; i < 2; i++)
   {
     delay(105);
     d = getData_Avg();
     if (sensor_error < Absolute_error)
       break;
   }
 }

 if (d == 0x7fffff)
 {
   Serial.println("Error: Failed to get data from HX711");
   return _d;
 }
 if (sensor_error > Absolute_error)
   Serial.println("Error Weight: " + String(toWeight(d)) + " +-" + String(sensor_error / Scale));
 return d;
}
```

delay_W()

```
/**
 * @brief Delay function with the ability to _detect the interruption signal and weight changes
 *
 * @param timeout     The maximum time to wait. if timeout = 0xffff, the function will wait indefinitely
 * @param time2listen The time to listen weight changes
 * @param error       Default is absolute error of the scale
 * @return byte       0: timeout, 1: weight changes, 2: interrupt signal
 */
byte delay_W(uint16_t timeout, uint16_t time2listen, uint16_t error)
{
 if (_detect == 1)
   return 1;

 unsigned long t = millis();
 unsigned long TIME_END = t + timeout;
 byte flag = 0;
 int32_t d = 0;
```

```
while (interrupt == prev_interrupt)
{
 if (timeout == 0xffff)
 {
  delay(time2listen);
  d = getData_(true);
 }
 else if (TIME_END > t + time2listen + 105)
 {
  delay(time2listen);
  d = getData_();
  t = millis();
 }
 else
 {
  delay(abs((int)(TIME_END - t)));
  t = TIME_END;
  break;
 }

 if (flag == 1)
  if (abs(d - Zero) > 5 * error)
  {
   _detect = 1;
   break;
  }
  else
   continue;

 if (abs(d - Zero) < error)
 {
  flag = 1;
  continue;
 }

 if (abs(_d - d) > error)
 {
  if (abs(d - getData_()) < 2 * error)
  {
   _detect = 1;
   break;
  }
 }
}

if (_detect == 1)
{
 timer = millis();
```

```
    return 1;
  }
  return (t == TIME_END) ? 0 : 2;
}
```