Bắt đầu:

**1.Khởi tạo các biến có kiểu cấu trúc dành cho các chức năng riêng:**

RTC\_HandleTypeDef hrtc;

ADC\_HandleTypeDef hadc1;

RTC\_HandleTypeDef hrtc;

SPI\_HandleTypeDef hspi1;

DMA\_HandleTypeDef hdma\_spi1\_tx;

DMA\_HandleTypeDef hdma\_spi1\_rx;

RTC\_TimeTypeDef sTime;

RTC\_DateTypeDef sDate;

TIM\_HandleTypeDef htim1;

TIM\_HandleTypeDef htim4;

DMA\_HandleTypeDef hdma\_tim1\_ch3;

**char** time[15];

**char** date[15];

**char** sensor[5]; //Các biến mảng để chứa các giá trị string đề suât ra LCD

**char** restminute[2];

**char** latelyoff[2];

**float** distance = 26; //Thời gian hiện tại khởi tạo bằng một giá trị tùy vào môi trường

**float** distance1; //lưu thời gian quá khứ

**float** distance2 = 0; //lưu thời gian hiện tại

**2. Khối lấy thông tin cảm biến siêu âm HRSR04:**

Thuật toán trong khối này cốt lõi sử dụng một biến numTicks để đếm thời gian mà một chu kì sóng siêu âm được gửi đi và quay về sau đó tính khoảng cách bằng công thức distance = (numTicks + 0.0f)\*2.8\*Speedofsound; có thể hiểu đơn giản là quãng đường bằng vận tốc \*thời gian (1 numTicks = 2.8s)

Cách để lấy dữ liệu là: gửi một xung 10us vào chân trigger của module cảm biến để kích hoạt cảm biến siêu âm:

HAL\_GPIO\_WritePin(Trigger\_GPIO\_Port, Trigger\_Pin, *GPIO\_PIN\_RESET*);

usDelay(3);

HAL\_GPIO\_WritePin(Trigger\_GPIO\_Port, Trigger\_Pin, *GPIO\_PIN\_SET*);

usDelay(10);

HAL\_GPIO\_WritePin(Trigger\_GPIO\_Port, Trigger\_Pin, *GPIO\_PIN\_RESET*);

Cùng lúc đó chúng ta xem xét tín hiệu chân Echo có bằng 0 hay không, nếu có thì đặt biến đếm numTicks = 0, còn nếu không thì biến numsTicks tăng lên 1 và chờ 2us

Sau đó biến distance sẽ được tính bằng công thức:

distance = (numTicks + 0.0f)\*2.8\*Speedofsound;

Cuối cùng chúng ta đã có được thông tin của cảm biến siêu âm

**3. Khối có nhiệm vụ xác định có người hay không**

-Khối này hoạt động bằng các xét xem giá trị cảm biến hiện tại có bằng giá trị cảm biến quá khứ không, nếu bằng thì xem như không có vật nào chuyển động từ đó suy ra không có người, trường hợp nếu có sự thay đổi thì ta xét xem giá trị thay đổi này có lớn hơn mức cho phép hay không, nếu sự thay đổi nhỏ thì vẫn bỏ qua và xem như chưa có người, nếu sự thay đổi này lớn(đáng kể) thì kết luận là có người hoạt động trong phòng

Code:

Ban đầu ta gán biến chứa giá trị khoảng cách hiện tại cho biến distance2, biến này cũng có cùng ý nghĩa với biến distance, mục đích là sử dụng biến distance2 để tránh ảnh hưởng tới biến distance phòng trường hợp sử dung biến distance cho mục đích khác

distance2 = distance;

Tiếp theo là một hàm if, else để thiết lập cho nhiệm vụ xác định có người hay không

**if**(((distance2-distance1)>=5)|(distance2<20))

{

**if** (nextFiveMinutes >= 60){

nextFiveMinutes -= 55 ;

};

human = 1;

nextFiveMinutes = sTime.Minutes;

nextFiveMinutes += 0b0101;

**sprintf**(restminute, "%02d'",nextFiveMinutes);

showzifustr(60, 135, restminute , BLACK, WHITE);

}

Hàm này xác định có người khi giá trị khoảng cách hiện tại khi distance2 – distance1 >=5 và khi phát hiện có vật thể quá gần cụ thể là < 20 (giá trị này có thể thay đổi tùy vào môi trường cụ thể khác nhau.

Thiết lâp biến human = 1, biến này có chức năng đại diện cho trạng thái có người (1) hoặc không có người(0)

Biến nextFiveMinutes này lưu giá trị của 5 phút sau kể từ phút hiện tại, cứ mỗi khi phát hiện có người thì biến này sẽ được cập nhật lên bằng câu lệnh:

nextFiveMinutes = sTime.Minutes;

nextFiveMinutes += 0b0101;

sau đó in giá trị ra màn hình LCD bằng câu lệnh:

**sprintf**(restminute, "%02d'",nextFiveMinutes);

showzifustr(60, 135, restminute , BLACK, WHITE);

hàm:

**if** (nextFiveMinutes >= 60) {

nextFiveMinutes -= 55 ;

};

có chức năng cài đặt lại giá trị phút về 5 khi nextFiveMinutes vượt qua ngưỡng 60 phút

Tiếp theo là hàm xác định không có người

**else**{

**if**(sTime.Minutes == nextFiveMinutes){

human = 0;

**sprintf**(timelately, "%02dh'",sTime.Hours);

showzifustr(80, 150, timelately , BLACK, WHITE);

**sprintf**(latelyoff, "%02d'",nextFiveMinutes);

showzifustr(100, 150, latelyoff , BLACK, WHITE);

};

Nếu không có người thì thì sẽ kiểm tra thời gian hiện tại có bằng thời gian nextFiveMinutes không, nếu không thì vẫn chưa đến lúc tắt đèn nên không có hành động gì diễn ra trong hàm, ngược lại nếu bằng **if**(sTime.Minutes == nextFiveMinutes) thì trang thái lúc này sẽ là không có người human = 0 sua đó là xuất trạng thái ra màn hình LCD.

**4.Khối điều khiển đèn dựa trên biến human.**

Ở đây có hai trạng thái, có người và không có người. Như đã cài đặt thì thanh ghi TIM-CCR3 chính là thanh ghi điều chỉnh độ rộng/hẹp của xung điều khiển đèn từ đó tăng hoặc giảm cường độ sáng của đèn cùng với đó ta có biến lux để lưu giá trọ cường độ ánh sáng được đọc từ chân ADC giá trị lux này sẽ giao động từ 0-4000 dựa theo độ sáng của môi trường thông thường độ sáng khoảng 2900 thì căn phòng sẽ sáng ở mức tốt nên em sẽ lấy giá trị trong khoảng 2900-2950 để làm mốc

**if**(human == 1){

**if**(lux < 2900)

{

**if**(TIM1->CCR3 <= 100){

TIM1->CCR3 += 5;

}

}

**if**(lux>2950){

**if**(TIM1->CCR3 > 0){

TIM1->CCR3 -= 5;

}

**if**(TIM1->CCR3 == 0){

TIM1->CCR3 = 0;

}

}

**if**((lux>=2900)&&(lux<=2950))

{

TIM1->CCR3 = TIM1->CCR3;

}

HAL\_TIM\_PWM\_Start(&htim1, TIM\_CHANNEL\_3);

Nếu trạng thái là có người thì thanh ghi xung này sẽ điều chỉnh độ sáng lên tư từ cho đến khi lux trong căn phòng đạt đến ngưỡng 2900(lux), nếu độ sáng trong căn phòng vượt quá 2950(lux) thì sẽ tự điều chỉnh giảm độ sáng đèn lại, nếu như đèn nằm trong khoảng 2900-2950 thì sẽ không tăng không giảm đọ sáng của đèn

**else**{

TIM1->CCR3 = 0;

HAL\_TIM\_PWM\_Start(&htim1, TIM\_CHANNEL\_3);

}

Ngược lại nếu là trạng thái không có người thì đèn sẽ tắt ngay lập tức

**5.Khối đọc giá trị ánh sáng lux**

HAL\_ADC\_Start(&hadc1);

HAL\_ADC\_PollForConversion(&hadc1, 20);

lux = HAL\_ADC\_GetValue(&hadc1);

**sprintf**(light, "%d (lux)", lux);

showzifustr(150, 140,light, BLACK, WHITE);

**sprintf**(control, "%d", TIM1->CCR3);

showzifustr(150, 155, control, BLACK, WHITE);

HAL\_Delay(1000);

KHởi động ADC và thiết lập chân adc ở chức năng PollForConversion sau đó dùng hàm HAL\_ADC\_GetValue(&hadc1) để đọc và lưu giá trị vào biến lux

**TỔNG QUAN VỀ GIAO TIẾP NGOẠI VI SPI (Serial Peripheral Interface)**

SPI thường được sử dụng giao tiếp với: Bộ nhớ EEPROM, LCD, RTC (Đồng hồ thời gian thực), IC âm thanh, các loại cảm biến như nhiệt độ và áp suất, thẻ nhớ như MMC hoặc thẻ SD hoặc các bộ vi điều khiển khác

**1.CẤU TẠO**

Bốn chân I/O dành riêng cho giao tiếp SPI với các thiết bị bên ngoài:

*a)SCK (Serial Clock):*

• Thiết bị Master tạo xung tín hiệu SCK và cung cấp cho Slave.

• Xung này có chức năng giữ nhịp cho giao tiếp SPI. Mỗi nhịp trên

chân SCK báo 1-bit dữ liệu đến hoặc đi.

→ Quá trình ít bị lỗi và tốc độ truyền cao.

*b)MOSI (Master Output Slave Input):*

- Dành cho việc truyền dữ liệu từ Master đến Slave. Chân MOSI ở Master sẽ kết nối đến chân MOSI ở Slave.

*c)MISO (Master Input Slave Output):* Dành cho việc truyền dữ liệu từ Slave đến Master. Chân MISO ở Master sẽ kết nối đến chân MISO ở Slave.

*d)CS/SS (Chip/Slave Select):* Chân CS được Master sử dụng để lựa chọn Slave cần giao tiếp để chọn Slave giao tiếp thiết bị Master chủ động kéo đường CS tương ứng xuống mức 0 (Low) nếu không các đường SCK, MOSI, MISO của

Slave sẽ không được kích hoạt

• Chân CS của vi điều khiển (Master) có thể được người dùng tạo bằng cách cấu hình 1 chân GPIO bất kỳ chế độ Output.

**2.Chế độ truyền:**

• Simplex (đơn công)

• Half duplex (bán song công)

• Full duplex (song công toàn phần).

Chế độ truyền xác định hướng của luồng tín hiệu giữa hai thiết bị được kết nối.

*CHẾ ĐỘ TRUYỀN CỦA SPI – FULL-DUPLEX COMMUNICATION*

Thanh ghi dịch của Master và Slave được liên kết qua hai đường một chiều: MOSI và MISO.

• Trong suốt quá trình giao tiếp SPI, data sẽ được dịch và đồng bộ trên sườn xung Clock được cung cấp bởi Master.

• Master truyền dữ liệu cần gửi đến Slave qua đường MOSI và nhận dữ liệu từ Slave qua đường MISO.

*CHẾ ĐỘ TRUYỀN CỦA SPI – HALF-DUPLEX COMMUNICATION*

• Thanh ghi dịch của Master và Slave được liên kết qua hai đường kết nối chéo và phải có điện trở (R).

• Trong suốt quá trình giao tiếp SPI, data sẽ được dịch và đồng bộ trên sườn xung Clock theo hướng truyền được chọn qua lại bởi cả Master và Slave

*CHẾ ĐỘ TRUYỀN CỦA SPI – SIMPLEX COMMUNICATION*

• Khi chỉ cần truyền dữ liệu (only Transmit) => Sử dụng 2 đường SCK, MOSI

• Master sẽ không bao giờ nhận gì từ Slave (Chỉ truyền)

• Chân CS của Slave phải được kéo xuống mức thấp

*CHẾ ĐỘ TRUYỀN CỦA SPI – SIMPLEX COMMUNICATION*

• Khi chỉ cần nhận dữ liệu (only Receive) => Sử dụng 2 đường SCK, MISO

**3.CÁCH TRUYỀN NHẬN DỮ LIỆU CỦA GIAO THỨC SPI**

Cấu tạo phần cứng SPI:

Master:

• Một thanh ghi dịch (Shift Register)

• Một mạch chốt dữ liệu (Latch)

• Một bộ tạo xung nhịp (Clock Generator)

Slave:

• Một thanh ghi dịch (Shift Register)

• Một mạch chốt dữ liệu (Latch)

Cả hai thanh ghi 8-bit (hoặc 16-bit) dịch được kết nối để tạo thành một vòng.

• Để bắt đầu quá trình truyền dữ liệu, Master sẽ cấu hình xung clock để tương ứng với tần số hỗ trợ bởi Slave, thường lên đến vài MHz.

• Nếu một Slave được Master chọn để giao tiếp, chân SS khi đó sẽ được ghi mức LOW.

1. Thiết bị Master bắt đầu việc trao đổi dữ liệu bằng cách truyền đi một Byte vào thanh ghi dịch của nó.

2. Sau đó, Byte dữ liệu sẽ được đưa sang Slave theo đường tín hiệu MOSI.

3. Slave sẽ truyền dữ liệu nằm trong thanh ghi dịch của chính nó ngược trở về Master thông qua đường tín hiệu MISO.

Lưu ý: Khi truyền một Byte dữ liệu thì sẽ bắt đầu bit có trọng số cao (bit số 7)

và kết thúc bằng bit có trọng số thấp nhất(bit số 0).

• Bằng cách này, dữ liệu của hai thanh ghi sẽ được trao đổi với nhau.

• Việc đọc và ghi dữ liệu vào Slave diễn ra cùng một lúc nên tốc độ trao đổi dữ liệu diễn ra rất nhanh.

→ Giao thức SPI là một giao thức rất có hiệu quả

4. Sau khi các bit trên thanh ghi dịch được đẩy ra hoặc đã nhận vào toàn bộ, cả Master và Slave đã hoàn tất việc trao đổi giá trị thanh ghi.

• Nếu nhiều data hơn cần được truyền đi, các thanh ghi dịch sẽ được load lại và quá trình trên được lặp lại.

5. Khi hoàn tất, Master ngừng việc đảo xung clock và ngừng chọn chân CS với Slave tương ứng.

**4. CHẾ ĐỘ HOẠT ĐỘNG CỦA GIAO THỨC SPI**

Mode 0: (CPOL = 0 và CPHA = 0).

• Truyền dữ liệu xảy ra khi gặp sườn lên của xung đồng hồ.

Mode 1: (CPOL = 0 và CPHA = 1).

• Truyền dữ liệu xảy ra khi gặp sườn xuống của xung đồng hồ.

Mode 2:   (CPOL = 1 và CPHA = 0

• Truyền dữ liệu xảy ra khi gặp sườn xuống của xung đồng hồ.

Mode 3:   (CPOL = 1 và CPHA = 1).

• Truyền dữ liệu xảy ra khi gặp sườn lên của xung đồng hồ.

Bảng giá linh kiện

|  |  |  |
| --- | --- | --- |
| Tên linh kiện | Số lượng | Thành tiền |
| STM32F401CDU6 | 1 | 100.000 VND |
| Cảm biến siêu âm HR-SR04 | 1 | 26.000 VND |
| LCD TFT 1.28 Round | 1 | 69.000 VND |
| Trở quang GL5516 | 1 | 2.000 VND |
| Trở, dây điện… | 1 | 20.000 VND |

|  |  |
| --- | --- |
| Tổng tiền | 217.000 VND |