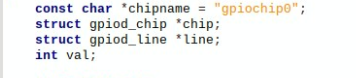
TH6.lab5

* ***Include các thư viện cần thiết***

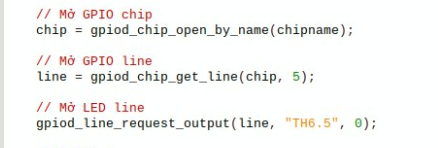


* ***Khai báo các biến cần dung***



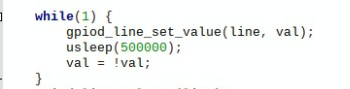
* ***Mở GPIO chip, line***

Ở đây bọn em sử dụng chân GPIO 5 và set trạng thái của LED ở mức 0 (đèn tắt)

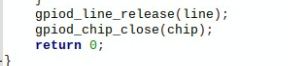


* ***Hàm while***

Hàm xử lý đổi trạng thái của đèn LED sau mỗi chu kỳ 0,5s

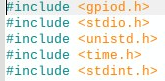


* ***Giải phóng chip và line***

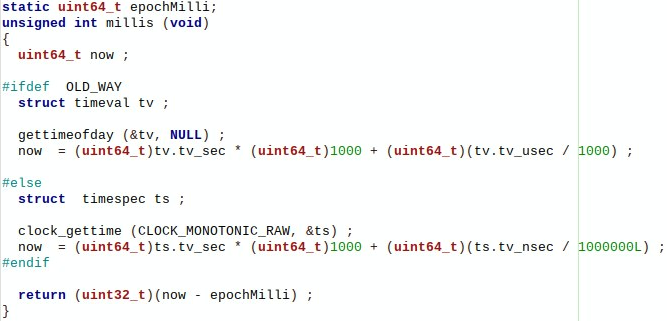


TH6.lab6

* ***Include các thư viện cần thiết, đặc biệt là gpiod.h***

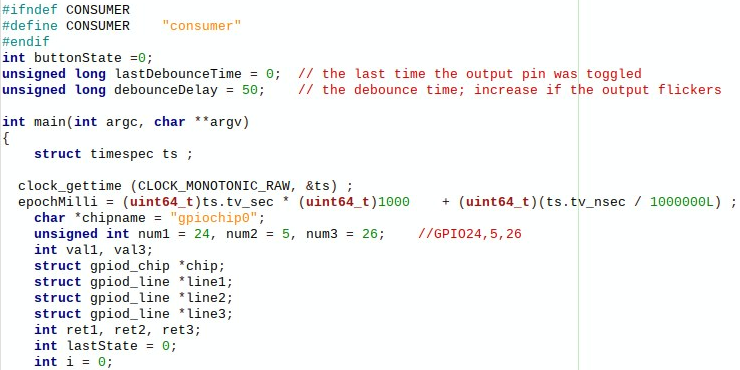


* ***Do tất cả các nút bấm đều bị nhiễu nên bọn em sử dụng hàm millis phục vụ cho việc debounce nút bấm. Hàm millis dựa theo thư viện wiringPi, trả về thời gian millisecond tính từ khi chạy chương trình***

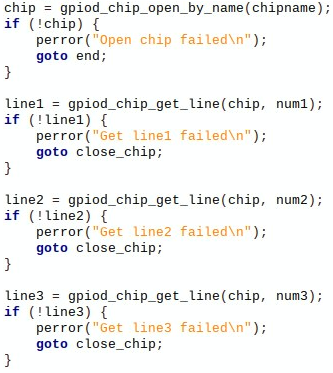


* ***Khai báo các biến cần thiết:***

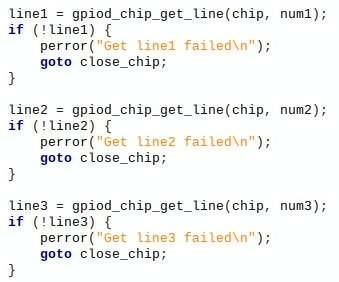
Ở đây bọn em sử dụng chân GPIO24 làm nút bấm, GPIO5 làm led và GPIO26 OUTPUT HIGH làm nguồn 3.3V cho nút bấm. (do lúc viết code thì không có đủ dây và dây không đủ dài để nối 3.3V từ Ras tới nút nên bọn em dùng GPIO26 đã được nối từ bài thực hành trước đó).



* ***Mở GPIO chip***

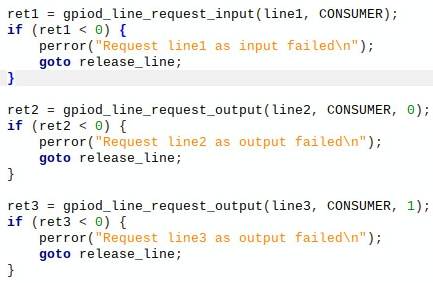


* ***Lấy con trỏ đối tượng GPIO 24, 5, 26***



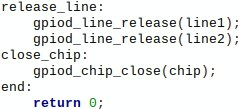
* ***Cấu hình GPIO 24 là input, GPIO5 và GPIO26 là output***

Ở đây bọn em set led (line2) ban đầu luôn ở 0 (tắt) và nguồn 3.3v (line3) luôn ở chế độ 1 (bật)

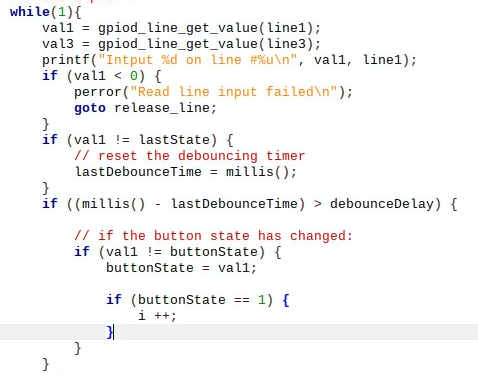


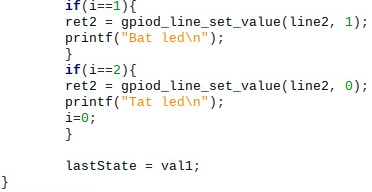
* ***Các nhãn cần thiết và để ở cuối hàm main***

Nếu việc mở chip, lấy con trỏ và cấu hình GPIO thất bại tương ứng sẽ goto những nhãn này.



* ***Hàm while(1)***





\*2 biến val1 và val3 để lần lượt lấy mức hoạt động của nút bấm và nguồn 3.3V

\*biến ret2 để set mức hoạt động của đèn led, 1 là bật còn 0 là tắt.

\*biến lastState để lưu trạng thái của nút nhấn trước đó. So sánh lần nhấn này với trạng thái trước đó nếu khác nhau thì tức là nút đã được nhấn hoặc được nhả. Nếu val1 mà bằng 1 thì là nhấn.

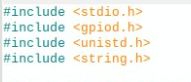
\*Quy trình thực hiện debounce cơ bản:

1. Kiểm tra trạng thái nút, nếu khác trạng thái trước thì tức là nút đã được nhấn hoặc nhả, lưu lại thời gian đổi trạng thái vào biến lastDebounceTime.

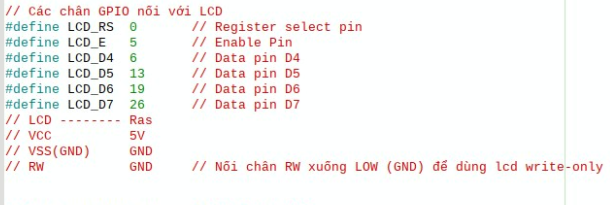
2. Lấy thời gian hiện tại trừ đi lastDebouceTime, nếu kết quả lớn hơn debounceDelay (50ms) thì tức là nút đã được nhấn mà không không bị nảy (nhiễu), lúc này có thể thực hiện công việc.

TH6.lab7

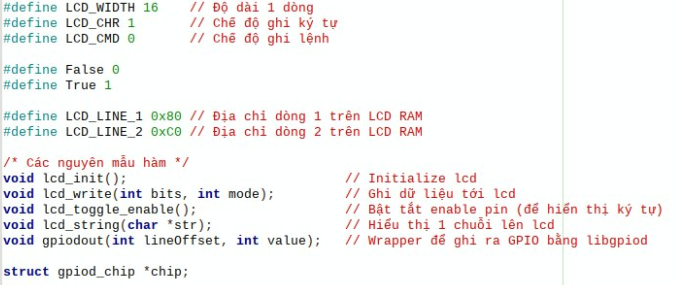
* ***Include các thư viện cần thiết***



* ***Cách nối các chân giữa GPIO với LCD***



* ***Khai báo các hàm và biến cần dung***



* ***Hàm void lcd\_init ()***

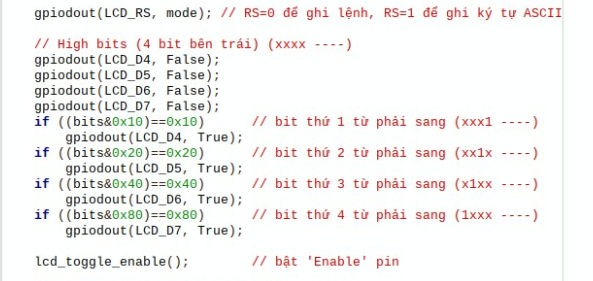
Dùng để ghi các lệnh khởi tạo LCD



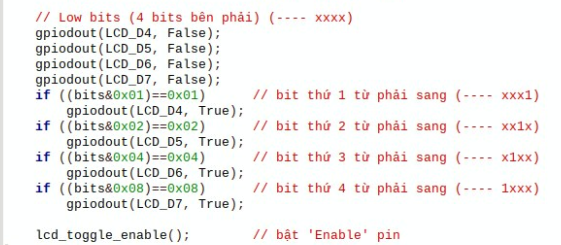
* ***Hàm void lcd\_write ()***

Dùng để ghi 1 byte tới LCD chế độ 4 bit

+ High bit (4 bit bên trái)

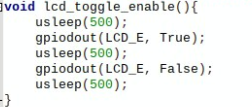


+ Low bit (4 bit bên phải)



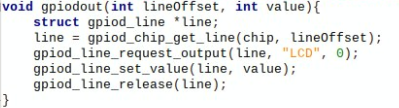
* ***Hàm lcd\_toggle\_enable()***

Dùng để bật tắt E pin, để thông báo kết thúc 1 lần



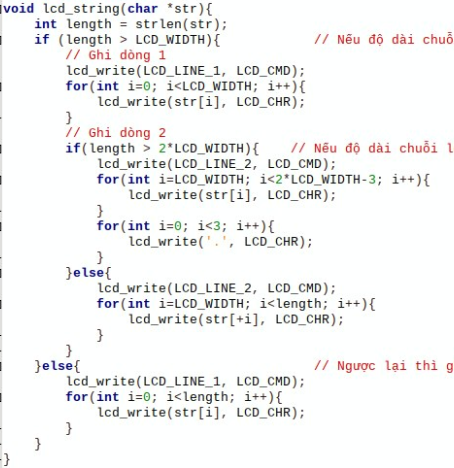
* ***Hàm gpiodout ()***

Dùng để ghi set line bằng số chân theo GPIO Broadcom (gpio offset) cho tiện.



* ***Hàm void lcd\_string ()***

Dùng để ghi 1 chuỗi ra màn hình LCD. Nếu độ dài chuỗi lớn hơn 1 dòng thì ghi thành 2 dòng, nếu độ dài chuỗi lớn hơn 2 dòng thì ghi 3 ký tự cuối của dòng 2 thành “…”



HÀM main()

* ***Mở GPIO chip***

Sau khi mở GPIO chip, nếu khi chạy chương trình có nhập tham số là 1 chuỗi thì in ra chuỗi đó lên LCD, không thì in ra dòng chữ “Hello world” sau đó nghỉ 3 giây rồi clear màn hình LCD



* ***Ghi 0 cho các chân GPIO đã mở và sau đó đóng GPIO chip***



TH6.lab8

* Tín hiệu UART luôn ở mức cao, khi có dữ liệu truyền đi thì start bit là LOW, theo sau đó là 8 bit dữ liệu: HIGH = 1 và LOW = 0, bit cuối cùng là stop bit = HIGH. UART luôn truyền đi từng byte một.
* Thứ tự bit truyền đi là LSB (Least Significant Bit), tức là gửi các bit theo thứ tự từ phải sang trái.
* Độ dài 1 bit được xác định bởi baud rate, với baud rate = 9600 bit/s thì độ dài 1 bit là 1s/9600 = ~104.16 us.
* Từ đó phải giải quyết 2 vấn đề:

1. làm thế nào để tính toán thời gian, từ đó để gửi/đọc chính xác từng bit?

2. làm thế nào để gửi và nhận đồng thời?

- Phương hướng thực hiện với baud rate = 9600:

+ Đối với TX:

1. Ghi chân TX = 0 (start bit), đợi 104us

2. Ghi TX tương ứng với các bit của dữ liệu bằng cách: TX = phép bitwise AND của data với 1 (00000001) = bit đầu tiên từ phải sang. Sau đó dịch bit của data sang phải 1 bit và lặp lại thực hiện bitwise + dịch bit 7 lần, giữa mỗi lần đợi ~104us

3. Ghi chân TX = 1 (stop bit)

+ Đối với RX:

1. Liên tục kiểm tra chân RX, nếu RX = 0 -> start bit, đợi 104us

2. Đọc RX, dịch bit RX sang trái i bit (i là thứ tự bit đến, tính từ 0). Thực hiện bitwise OR với biến data (khởi tạo bằng 0), sau đó lưu vào biến data. Lặp lại thêm 7 lần để lấy đủ 8 bit, mỗi lần đợi 104us

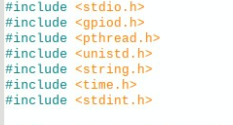
3. Lặp lại cho tới khi data = 10 (tức byte tới là ký tự '\n')

+ Đối với việc đợi thời gian:

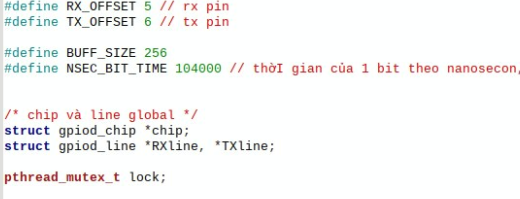
1. Lấy thời gian hiện tại tính bằng ns (nanosecond) lưu vào 1 biến last
2. Sử dụng hàm while để liên tục kiểm tra xem từ lúc lưu biến last tới giờ đã đủ thời gian cần đợi chưa, rồi mới tiếp tục

Bởi vì sử dụng soft timer, nên độ chia nhỏ nhất tối thiểu bằng 1 chu kỳ CPU, có thể bị ảnh hưởng bởi thời gian thực hiện lệnh, thời gian get/set GPIO, hoặc bởi lập lịch CPU. Từ đó nên kết quả khi gửi và nhận dữ liệu có thể sai sót do lệch thời gian của bit.

- ***Include các thư viện cần thiết***

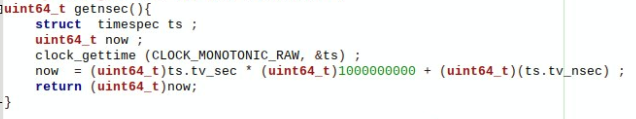
******

* ***Khai báo biến và các chân GPIO***

******

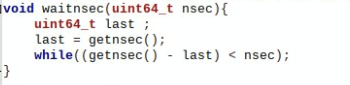
* ***Hàm getnsec ()***

Dùng để lấy thời gian hiện tại theo nanosecond



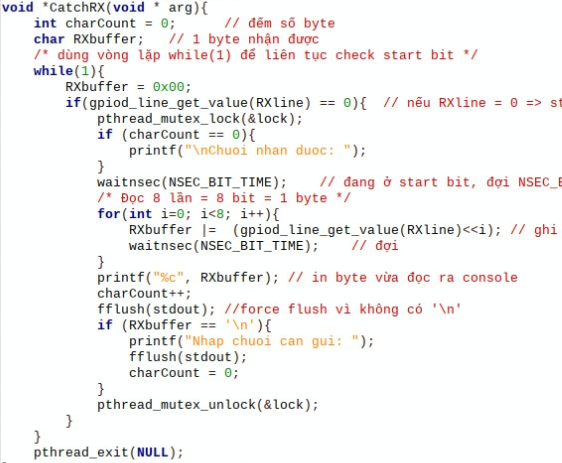
* ***Hàm waitnsec ()***

Dùng vòng lặp while để delay một khoảng thời gian bằng nsec, như đã giải thích ở trên.



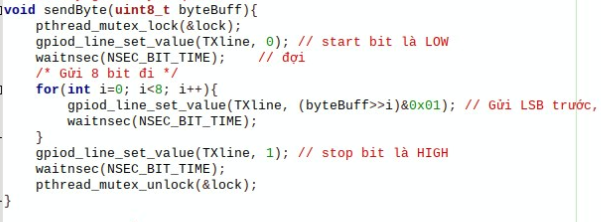
* ***Hàm \*CatchRX ()***

Sử dụng thread để bắt dữ liệu từ RX



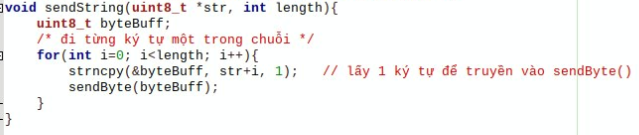
* ***Hàm sendbyte ()***

Dùng để gửi 1 byte qua TX



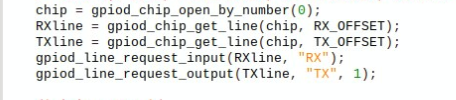
* ***Hàm sendString ()***

Hàm gửi 1 chuỗi, là 1 wrapper quanh hàm sendByte()

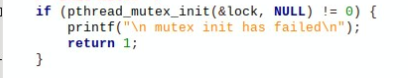


* ***Trong hàm main ()***

***+ Khởi tạo line, RX là input và TX output***



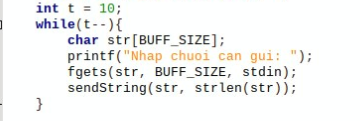
+ ***Init mutex***

******

***+ Tạo RX thread***

******

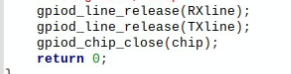
***+ Thực hiện gửi 10 lần***

******

***+ Đóng thread***

******

***+ Đóng line, chip***

******