**بخش 1 : LCD**

**void send\_4bit(unsigned char data)**

برای ساختار یافته کردن کد در ابتدا این تابع را تعریف می کنیم که مقدار هر کدام از پین های داده LCD را با توجه داده ورودی تنظیم می کنیم.

**void lcd\_write(unsigned char data)**

بدلیل اینکه در مد 4 بیت میخواهیم LCD را راه بیندازیم باید داده ورودی را به رو دو بخش LSB و MSB تقسیم می کنیم و در دو مرحله این دو رو به LCD ارسال کنیم.

برای فرستادن به LCD ابتدا باید پایه enable را یک کنیم, سپس با استفاده از تابع check\_LSB\_MSB در ابتدا پین های LCD را ریست کرده و بخش MSB را ارسال کرده و پایه enable را ریست می کنیم. و بار دیگر همین روند را برای فرستادن بخش LSB نیز انجام می دهیم.

برای تمیز کردن کد و ست و ریست کردن پایه enable, تابع blink\_En را می نویسیم که در ابتدا پایه enable را 1 و سپس 0 میکند.

**void lcd\_command(unsigned char command) :**

همانطور که می دانیم برای فرستادن دستور به LCD باید مقدار پایه rs صفر باشد.

پس در این تابع ابتدا پایه rs را صفر کرده و سپس با استفاده از تابع lcd\_write , دستور را به LCD می فرستیم.

**: void lcd\_data(unsigned char data)**

همانطور که می دانیم برای فرستادن داده به LCD باید مقدار پایه rs یک باشد.

پس در این تابع ابتدا پایه rs را یک کرده و سپس با استفاده از تابع lcd\_write , دیتا را به LCD می فرستیم.

**void LCD\_Init(void)**

در این تابع میخواهیم LCD را راه اندازی کنیم که باید دستورات زیر را برای ان ارسال کنیم.

(Function set)0x28 : نشان می دهد که در مد 4 بیتی کار می کنیم, LCD دو خط دارد و هر خانه ی آن 5\*8 پیکسل می باشد.

(Entery mode set)0x06 : نشان دهنده جهت حرکت کرسر می باشد که به سمت راست حرکت می کند و با چاپ هر کاراکتر ادرس DDRAM افزایش پیدا میکند و همچنین صحفه شیفت پیدا نمیکند.

(Display control)0x0C : این دستور برای روشن شدن LCD و خاموش بودن cursor و چشمک نزدن آن استفاده می شود.

همچنین از تابع lcd\_clear هم استفاده می کنیم تا lcd پاک شود و کرسر به خانه اول باز گردد.

**void LCD\_PutChar(unsigned char data)**

برای این که هنگامی که خط اول پر می شود به خط دوم برویم, یک کانتر تعریف می کنیم که اگر کانتر کمتر از 16 بود فقط کافی است ورودی را چاپ کنیم.

اگر این کانتر برابر 16بود یعنی خط اول پر شده است, پس از تابع LCD\_SetCursor استفاده می کنیم و کرسر را به خط دوم منتقل می کنیم.

حالت بعدی اگر این کانتر برابر 32 باشد یعنی LCD پر شده است, پس باید به خط اول برگردیم و همچنین این کانتر را نیز صفر کنیم.

**void LCD\_PutString(char \*str)**

همانطور که می دانیم هر رشته از حروف در اخر خود یک ‘\0’ دارند که نشان دهنده این است که رشته به انتها رسیده است.

پس با استفاده از یک حلقه پوینتر ورودی را دی رفرنس کرده و با استفاده از تابع LCD\_PutChar آن را روی LCD چاپ می کنیم.

**void LCD\_SetCursor(unsigned char row, unsigned char col)**

با توجه به دیتاشیت آدرس اولین خانه سطر اول از 0x00 شروع شده و ادرس اولین خانه سطر دوم نیز از 0x40.

حال می توان از دستورالعمل set ddram address استفاده کرد و ادرس کرسر را تغیر داد.

در اینجا یک سوییچ روی row قرار دادیم که اگر row برابر 1 بود باید 0x80 را با Col-1 جمع می کنیم تا به خانه مورد نظر در سطر اول برسیم و همچنین برای سطر دوم نیز با عدد 0XC0 جمع کنیم.

همچنین همانطور که قبلا گفته شده است, کانتر را باید تغییر دهیم تا شمارش کاراکتر های چاپ شده به هم نخورد.

**void LCD\_CreateChar(uint8\_t Location,unsigned char data[]) :**

در ابتدا باید با استفاده از دستور set CGRAM address , ادرس خانه ای که میخواهیم کاراکتر جدید در آن نوشته شود را مشخص کنیم.

دقت شود که فقط 8 کاراکتر جدید می توانیم اضافه کنیم که ادرس آن ها می تواند از 0x00 تا 0x07 باشد.

سپس با استفاده از یک حلقه for اطلاعات مربوط به هر سطر را بصورت دیتا به LCD می فرستیم.

با توجه به دیتاشیت با اجرا کردن CGRAM data write , کانتر آدرس رم افزایش یک واحدی خواهد داشت و لازم نیست در هر مرحله آدرس CGRAM را افزایش دهیم و سپس داده را بفرستیم.

**void LCD\_PutCustom(uint8\_t Location) :**

برای اینکه بعد از ذخیره کردن کاراکتر جدیدی بتوانیم آن را نمایش دهیم باید از دستور set ddram address استفاده کنیم که این دستور در تابع LCD\_SetCursor, به کار گرفته شده است .

برای ذخیره کردن مکان کرسر در هر لحظه نیز تو متغییر row\_curosr و col\_cursor را تعریف می کنیم که مقدار آن ها با توجه به اینکه چند کاراکتر در LCD نوشته شده است, تغییر می کند.

**بخش دوم : Stepper Motor**

برای درایو موتور از ماژول uln2003 استفاده می کنیم.

در این ماژول با HIGH شدن ورودی, خروجی متناظر با آن low می شود, پس قسمت common موتور را به vcc وصل می کنیم و برای تحریک شدن هر سیم پیج باید ورودی متناظر با آن در uln2003 را HIGH کنیم.

**void stepper(int direction) :**

در این تابع موتور یه اندازه یک استپ می چرخد.

اگر direction 0 باشد, ساعتگرد و اگر برابر 1 باشد پادساعتگرد میچرخد.

در اینجا متغیری به نام Steps تعریف می کنیم که نشان دهنده این است که در مرحله بعدی کدام یک از مراحل تحریک سیم پیچ ها انجام شود.

برای جهت ساعتگرد این متغیر باید اضافه شود تا به مرحله بعدی برویم و برای پادساعتگرد نیز باید این متغیرکم شود تا جهت حرکت عوض شود.

متوجه شدیم که با توجه به جدولی که در ادامه آورده شده است, اگر در 0 درجه باشیم و بخواهیم پادساعتگرد بچرخیم , مرحله 8 باعث تغییر زاویه موتور نمیشد , پس اگر در زاویه صفر باشیم و بخواهیم در پادساعتگرد بچرخیم باید به مرحله 7 برویم.

برای مقدار دهی به پین های به پین های موتور نیز از تابع setMotor استفاده می کنیم که وضعیت 0 یا 1 بودن پین ها را در ورودی اش مشخص می کنیم.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| مرحله | A | B | C | D |
| 1 | 1 | 0 | 0 | 0 |
| 2 | 1 | 1 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 |
| 4 | 0 | 1 | 1 | 0 |
| 5 | 0 | 0 | 1 | 0 |
| 6 | 0 | 0 | 1 | 1 |
| 7 | 0 | 0 | 0 | 1 |
| 8 | 1 | 0 | 0 | 1 |

\*\* همان طور که قبلا اشاره شد , HIGH شدن ورودی uln2003 , به معنای تحریک شدن سیم پیچ مربوطه می باشد.

**void stepper\_degree(float teta) :**

برای ذخیره کردن زاویه موتور متغیر current\_position را تعریف می کنیم که عددی صحیح می باشد و با ضرب کردن در 0.9 زاویه موتور بدست می آید.

ابتدا باید ببینیم با توجه به زاویه فعلی موتور و زاویه ای که میخواهیم به آن بریم, چند درجه حرکت لازم است.

برای انجام دادن کمترین حرکت , اگر این تغییر زاویه بیشتر از 180 درجه باشد, کافی است به اندازه teta\_to\_move – 360 بچرخیم.

به طور مثال اگر بخواهیم از 10 درجه به 200 درجه برویم , کافی است به 170-) ( ( 200 – 10) – 360 ) بچرخیم که علامت منفی نشان دهنده پاد ساعتگرد بود حرکت است.

در حالت دیگر اگر بخواهیم از 200 درجه به 10 درجه برویم کافی است به اندازه teta\_to\_move + 360 درجه بچرخیم. ( (10 – 200) + 360 ) = +170

برای بدست آوردن اینکه چند استپ باید بچرخیم, زاویه را باید در ابتدا بدلیل اینکه هر استپ 0.9 درجه است , به 0.9 تقسیم کنیم و آنرا گرد کنیم.

اگر تعداد استپ ها عدد منفی باشد, متغیر direction را برابر 1 قرار می دهیم و متغیر step\_to\_move را منفی می کنیم.

برای اینکه زمان بندی دقیقی برای حرکت موتور داشته باشیم, تایمر 1 را روی 500 میلی ثانیه تنظیم می کنیم که در اینترآپت مربوط به آن, با توجه به اینکه جهت حرکت چگونه است , تا زمانی که steps\_to\_move به صفر نرسیده است, از تابع stepper استفاده می کنیم و موتور را یک استپ می چرخانیم.

متغیر current\_position هم با توجه به جهت حرکت باید کم و یا زیاد کنیم تا زاویه موتور را ذخیره کنیم.

\*\* با توجه به اینکه کلاک تایمر ها روی 8MHz تنظیم شده است برای داشتن اینترآپت های 500 میلی ثانیه ای باید تنظیمات زیر در نرم افزار stmcube انجام شود.

PreScaler = 63

CounterPeriod = 62499

با توجه به این که اینتراپت چرخیدن موتور باید اولویت بیشتری داشته باشد, Preemption اینتراپت های مختلف ( اینتراپت نوشتن زاویه فعلی روی LCD و اینتراپت زده شدن یکی از دکمه های کیپد و اینتراپت های مربوط SYSTEM tick timer ) به صورت زیر مقدار دهی شود.

اولویت اول : TM1 Update interrupt ( مربوط به چرخش موتور ) 🡺 Preemption = 0

اولویت دوم : Preemption = 1 🡺 system tick timer

اولویت سوم : TIM2 global interrupt ( مربوط به نوشتن LCD ) = 2 🡺 Preemption

اولویت چهارم : EXTI line[15:10] ( مربوط به کیپد ) 🡺 3 Preemption

\*\* تایمر 2 نیز که مربوط به LCD می باشد نیز همانند TIM1 طوری تنظیم شده است که هر 500 میلی ثانیه اینترآپت داشته باشیم.

**توضیحات کلی کد :**

در ابتدا چون کاربر زاویه ای را وارد نکرده است در menu1 هستیم و زاویه را صفر نشان می دهد.

با زدن دکمه = کیپد و آمدن اینتراپت مربوطه, متغیر flag\_interupt را یک می کنیم که نشان دهنده آن است که کلیدی زده شده است.

حال در حلقه while چک می کنیم که اگر در menu1 هستیم ( menu1\_state = 1 ) و اینتراپت آمد , عملیات اسکن کیپد را انجام می دهیم و اگر کلید زده شده برابر = بود , به menu2 برای دریافت زاویه از کاربر برویم.

بار دیگر اگر اینتراپت کیپد آمد و در menu2 ( menu1\_state = 0 ) باشیم , دوباره عملیات اسکن کیپد را انجام می دهیم و اگر کلید زده شده برابر = بود, زاویه وارد شده توسط کاربر را به تابع stepper\_degree پاس می دهیم و دوباره به menu1 که مخصوص نشان دادن زاویه فعلی موتور است, بر می گردیم.

اگر در menu2 بودیم و کلید زده شده چیزی غیر از = بود, آن را روی LCD چاپ می کنیم.

برای ذخیره کردن عددی که کاربر وارد می کند متغیر number\_pressed را تعریف می کنیم.

در این جا دو حالت به وجود می آید.

اگر کلید زده شده عدد باشد , ما باید number\_pressed را در 10 ضرب و با عدد زده شده جمع کنیم,

و اگر کلید زده شده برابر . باشد , ما متغیر dot\_pressed را تعریف می کنیم که نشان دهنده آن است که ممیز زده شده است. حال در مرحله بعد و با وارد کردن عدد توسط کاربر , باید این عدد وارد شده را بر 10 تقسیم و با number\_pressed جمع کنیم و dot\_pressed را دوباره صفر کنیم تا در حالت های بعدی عوض کردن زاویه مشکلی پیش نیاید.

**void menu1() :**

در این تابع ابتدا با توجه به current\_position , زاویه فعلی موتور را بدست می آوریم و در 10 ضرب می کنیم تا اگر عددی اعشاری بود به عددی صحیح تبدیل شود.

سپس با استفاده از این نکته که باقیمانده تقسیم یک عدد بر 10, یکان آن عدد را به ما می دهد در هر مرحله یکان را بدست می آوریم و عدد را به 10 تقسیم می کنیم تا باقیمانده در مرحله بعدی به دهگان این عدد برسیم.

\*\* با توجه به دیتاشیت نیز برای نوشتن علامت درجه روی LCD نیز باید 0XDF را بصورت دیتا روی پایه های LCD بفرستیم.