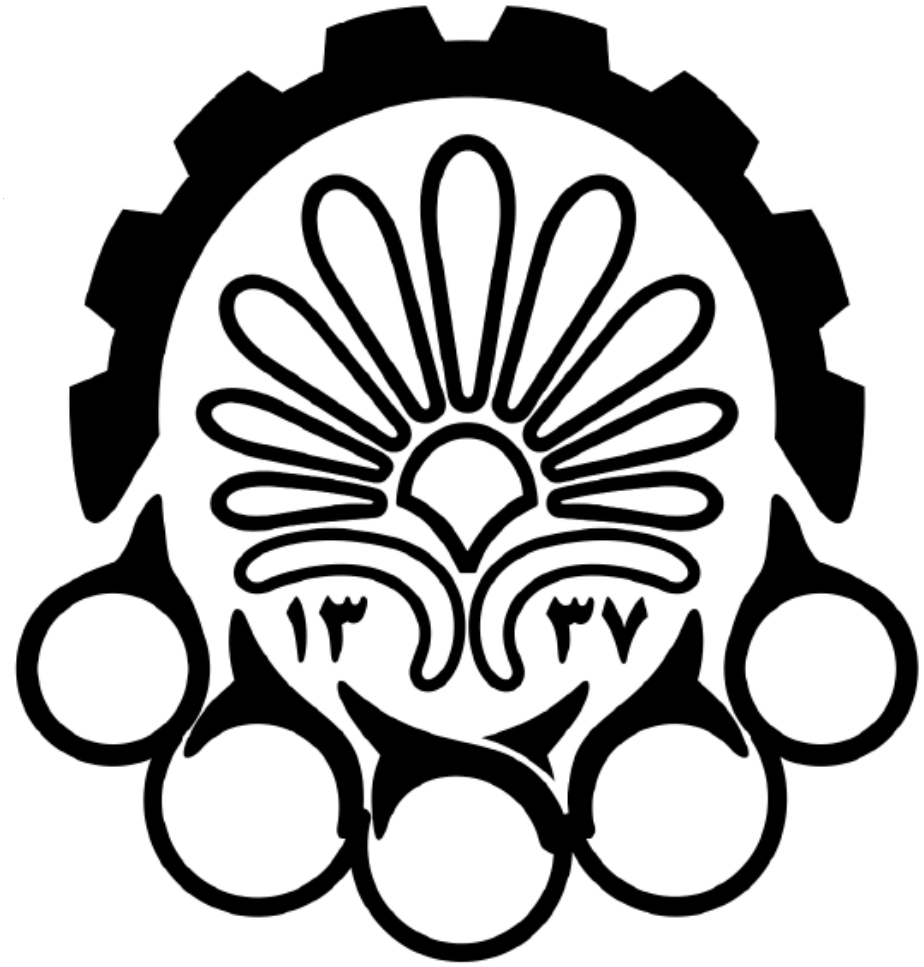


دانشگاه صنعتی امیرکبیر

(پلی تکنیک تهران)

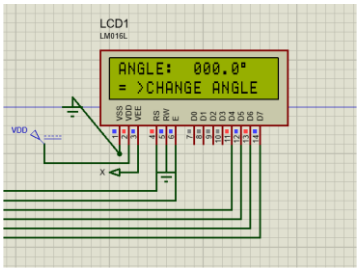
آزمایشگاه ریزپردازنده

گزارش آزمایش دوم

سهیل داوودی ۹۵۲۳۰۴۱

محمد جواد رنجبر ۹۵۲۳۰۴۸

بخش 1 : LCD



void send_4bit(unsigned char data)

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

void lcd_write(unsigned char data)

بدلیل اینکه در مد ۴ 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 تمیزکردن کد و ست و ریست کردن پایه enable, تابع blink_En را می نویسیم که در ابتدا پایه enable را ۱ و سپس ۰ میکند.

void lcd_command(unsigned char command):

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

پس در این تابع ابتدا پایه rs را صفرکرده و سپس با استفاده از تابع lcd_write , دستور را به LCD می فرستیم.

void lcd_data(unsigned char data)

همانطورکه 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.

0x28(Function set) : نشان می دهدکه در مد ۴ بیتی کار می کنیم, LCD دو خط دارد و هر خانه ی آن 5*8 پیکسل می باشد.

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

0x0C(Display control) : 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 وکرسر به خانه اول بازگردد.

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 اگرکانترکمتر از 16 بود فقط کافی است ورودی را چاپ کنیم.

اگر این کانتر برابر 16 بود یعنی خط اول پر شده است, پس از تابع LCD_SetCursor استفاده می کنیم وکرسر 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 صفرکنیم.

void LCD_PutString(char *str)

همانطورکه می دانیم هر رشته از حروف در اخر خود یک '\0' دارندکه 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 ۱ 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 همانطورکه 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 ۸ کاراکتر جدید می توانیم اضافه کنیم که ادرس آن ها می تواند از 0x00 تا 0x07 باشد.

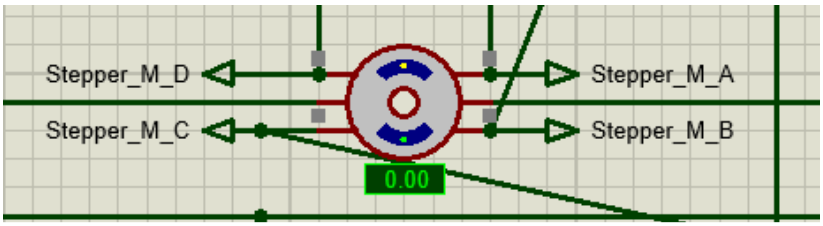
سپس با استفاده از یک حلقه for اطلاعات مربوط به هر سطر را بصورت دیتا به LCD می فرستیم.

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 کارگرفته 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 چندکاراکتر 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 پیچ 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 ۰ باشد, ساعتگرد و اگر برابر ۱ 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.

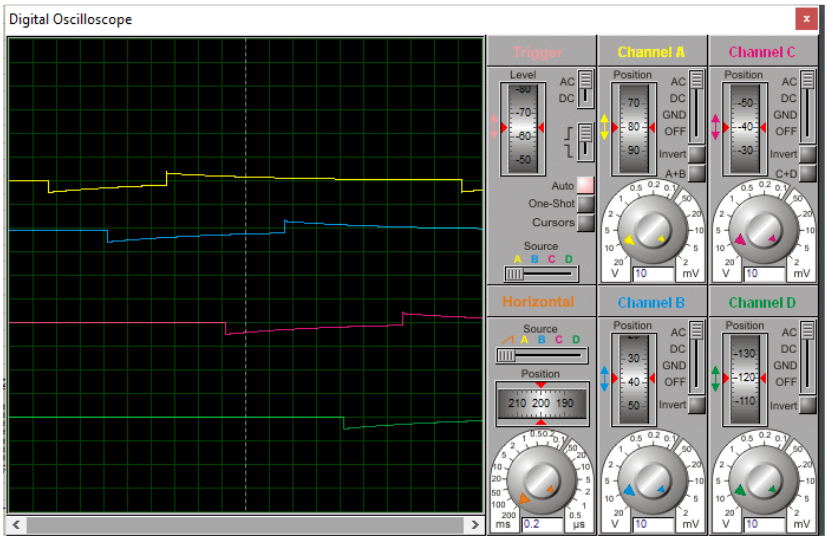
متوجه شدیم که با توجه به جدولی که در ادامه آورده شده است, اگر در ۰ درجه باشیم و بخواهیم پادساعتگرد بچرخیم , مرحله ۸ باعث تغییر زاویه موتور نمیشد , پس اگر در زاویه صفر باشیم و بخواهیم در پادساعتگرد بچرخیم باید به مرحله ۷ برویم.

برای مقدار دهی به پین های به پین های موتور نیز از تابع setMotor استفاده می کنیم که وضعیت ۰ یا ۱ بودن پین ها را در ورودی اش مشخص می کنیم.

مرحله	A	B	C	D
۱	1	0	0	0
۲	1	1	0	0
۳	0	1	0	0
٤	0	1	1	0
٥	0	0	1	0
٦	0	0	1	1
۷	0	0	0	1
۸	1	0	0	1

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

سوال ۱ :



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 ۰/۹ 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 ۱۸۰ درجه باشد, کافی است به اندازه 360 – teta_to_move بچرخیم.

به طور مثال اگر بخواهیم از ۱۰ درجه به ۲۰۰ درجه برویم , کافی است به ۱۷۰-((360 – (10 – 200)) بچرخیم که علامت منفی نشان دهنده پاد ساعتگرد بود حرکت است.

در حالت دیگر اگر بخواهیم از ۲۰۰ درجه به ۱۰ درجه برویم کافی است به اندازه 360 + teta_to_move درجه بچرخیم. = (360 + (10 – 200)) +170

برای بدست آوردن اینکه چند استپ باید بچرخیم, زاویه را باید در ابتدا بدلیل اینکه هر استپ ۰/۹ درجه است , به ۰/۹ تقسیم کنیم و آنرا گرد کنیم.

اگر تعداد استپ ها عدد منفی باشد, متغیر direction را برابر ۱ قرار می دهیم و متغیر step_to_move را منفی می کنیم.

برای اینکه زمان بندی دقیقی برای حرکت موتور داشته باشیم, تایمر ۱ را روی ۵۰۰ 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 بایدکم و یا زیادکنیم تا زاویه موتور را ذخیره کنیم.

سوال ۲ :

به جنس موتور ولختی ان وابسته است و یا به rpm-max و rpm-min وابسته است.

rpm_min/400<x<rpm_max/400

سوال ۳ :

** با توجه به اینکه کلاک تایمر ها روی 8MHz تنظیم شده است برای داشتن اینترآپت های ۵۰۰ 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 : system tick timer ➔ Preemption = 1

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

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

** تایمر ۲ نیزکه مربوط به LCD می باشد نیز همانند TIM1 طوری تنظیم شده است که هر ۵۰۰ 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 اگرکلید زده شده برابر = بود , به menu2 برای دریافت زاویه ازکاربر برویم.

بار دیگر اگر اینتراپت کیپد آمد و در menu2 (menu1_state = 0) باشیم , دوباره عملیات اسکن کیپد را انجام می دهیم و اگرکلید 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 وکلید 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.

اگرکلید زده شده عدد باشد , ما باید number_pressed را در ۱۰ ضرب و با عدد زده شده جمع کنیم,

و اگرکلید 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 واردکردن عدد توسط کاربر , باید این عدد وارد شده را بر ۱۰ تقسیم و با number_pressed جمع کنیم و dot_pressed را دوباره صفرکنیم 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 ۱۰ 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 ۱۰, یکان آن عدد را به ما می دهد در هر مرحله یکان را بدست می آوریم و عدد را به ۱۰ 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.