به نام خدا

اعضا : زهرا قربانی_سارا سلطانی

توضیحات :

قسمت الف) وقفه های تودرتو:

همانطور که میدانیم در avr به صورت خودکار هنگام رخداد یک وقفه و قبل از ورود به handler مربوط به آن، فلگ مربوط به interrupt در رجیستر وضعیت غیرفعال شده و بدین صورت امکان رخداد وقفه وجود ندارد. این غیرفعال بودن تا اتمام handler و اجرا شدن دستور RETI (که به صورت خودکار در انتهای وقفه انجام میشود) ادامه دارد. با اجرا شدن دستور RETI فلگ مربوط به interrupt در رجیستر وضیت فعال شده و دوباره امکان رخداد وقفه ها فراهم میشود.

پس در avr به صورت اولیه و دیفالت وقفه های تودرتو نداریم. این امکان(غیرفعال کردن وقفه های تودرتو) مزیت هایی دارد. بدین صورت حین اجرای وقفه های حساس به سطح مشکلاتی مانند وقفه های بی نهایت نداریم.

حال برای اینکه بتوانیم وقفه های تودرتو داشته باشیم امکانی فراهم شده است. که باید توسط برنامه نویس استفاده و مدیریت شود و باگ های احتمالی تشخیص و رفع شوند.
یکی از ساده ترین راه ها برای داشتن وقفه های تودرتو این است که بعد از اینکه وارد handler یک وقفه شدیم، فلگ interrupt را در رجیستر وضعیت فعال کنیم. برای این کار باید دستور sei() را در ابتدای آن handler فراخوانی کنیم بدین صورت حین اجرای این وقفه امکان رخداد وقفه های دیگر نیز فراهم میشود. اما نکته ای در استفاده از این روش وجود دارد. از آنجا که فعال سازی وقفه درون تابع انجام میشود، در کد کامپایل شده دستوراتی قبل از آن اجرا میشوند پس حین اجرای آن دستورات همچنان وقفه غیرفعال است.

برای اینکه مشکل روش قبل را حل کنیم کامپایلر امکاناتی را در اختیارمان قرار داده است. اگر در تعریف تابع به صورت زیر عمل کنیم به صورت خودکار دستور sei() قبل از هر دستور دیگری مربوط به handler اجرا میشود و بدین صورت وقفه های تودرتو داریم.

```
ISR(XXX_vect, ISR_NOBLOCK)
{
    instructions//
}
```

اما از آنجایی که ما میخواهیم ورود به یک هندلر را از طریق نمایش بر lcd ملاحضه کنیم نیاز است که اندکی درون یک هندلر بمانیم و پس از آن امکان اجرای وقفه های دیگر را فراهم کنیم پس از روش فراخوانی sei()استفاده میکنیم تا بتوانیم فرصتی برای نمایش بر روی Lcd داشته باشیم و امکان استفاده از ISR_NONBLOCK میسر نیست.

از طرف دیگر باید توجه داشته باشیم اگر فلگ interrupt را در وقفه های حساس به سطح فعال کنیم وقفه مدام تکرار شده و وارد یک لوپ بی نهایت می شود.پس باید وقفه های مورد نظر حساس به لبه باشند (در این مورد نیز از تایمرهای صفر و یک استفاده شده که حساس به لبه هستند و وقفه ها اورفلو شدن این تایمرهاست).

در نهایت در روند اجرای این کد از lcd برای بررسی مراحل استفاده کرده ایم که هنگامی که در تابع main قرار داریم و به عبارتی برنامه اصلی درحال اجراست عبارت in main بر روی lcd نمایان میشود. و هر گاه وارد هندلر وقفه i ام شدیم به صورت handler i آن را نمایش میدهیم و از آنجا که وقفه های تودرتو مجاز هستند، مشاهده میکنیم که وقفه ها به صورت تودرتو و بی نهایت اجرا میشوند و فرصتی برای اجرای برنامه اصلی باقی نمیماند و اجرای وقفه ها به صورت یکی در میان را شاهد هستیم.

قسمت ب) وقفه های تو در تو فقط تا سه سطح :

برای شمارش وقفه های تودرتو و چک کردن اینکه از سه سطح فراتر نرود از یک متغیر global استفاده میکنیم:

```
;Volatile unsigned int interrupt_count
```

در بدنه تابع main این متغیر را مقداردهی کرده ومقدار صفر را به آن میدهیم.

حالا بعد از رخداد وقفه درست در ابتدای handler مربوط به آن وقفه ابتدا مقدار این متغیر را یکی زیاد کرده و سپس در صورت اینکه برابر با سه نیست(کمتر از سه است)، فلگ مربوط به interrupt در رجیستر وضعیت را ست میکنیم. بدین صورت اگر تعداد وقفه ها کمتر از سه است اجازه رخداد وقفه های بیشتر وجود دارد اما اگر وقفه در حال اجرا وقفه سوم تودرتو است دیگر اجازه رخداد وقفه داده نمیشود.

حال در انتهای handler هر وقفه باید مقدار این متغیر را یکی کم کنیم چرا که یکی از وقفه ها تمام شده است و یک سطح کم میشود.

نکته ای را باید اینجا ذکر کرد: از آنجایی که میخوایم تعداد وقفه ها را شمارش کنیم و در صورتی که وقفه های تودرتو در سطح کمتر از سه است اجازه وقفه ی تودرتوی دیگر را بدهیم، در اینجا امکان استفاده از روشی که در قسمت قبل بیان شد(استفاده از ISR_NOBLOCK) وجود ندارد. روش فراخوانی sei() در ابتدای تابع handler را در این قسمت استفاده میکنیم که تا زمانی که هنوز sei() را اجرا نکرده ایم (افزایش کانتر و چک کردن مقدار کانتر) امکان رخداد وقفه وجود ندارد.

مجددا در روند اجرای این کد از lcdبرای بررسی مراحل کار استفاده کریم.مشابه قسمت الف هنگامی که تابع مین در حال اجراست عبارت in_main بر روی lcd نمایان میشود.و زمانی که برنامه به سمت اجرای وقفه های تودرتو میرود مقدار متغیر counter بر روی lcd نمایان میشود که نشان دهنده عمق وقفه های تودرتو است.این مقدار هیچگاه بیشتر از سه نخواهد بود چون با هربار اجرای وقفه چک میشود که این مقدار کمتر از سه باشد.(این عبارات هربار با کال کردن تابع lcd_print بر روی lcdنمایان میشوند.)

قسمت ج) وقفه های تودرتو فقط تا سه سطح و با رعایت اولویت :

این قسمت را میتوانیم در ۲ بخش مجزا بررسی کنیم:

بخش اول حالتی است که دو وقفه با هم برسند که در این حالت خود AVRبراساس آدرس ISR ها که در وکتور مربوطه ذخیره شده اند به وقفه ها اولویت میدهد(اگر وقفه ای در آدرس پایینتر قرار گرفته باشد اولویت بالاتری دارد و همان وقفه اجرا میشود.)همچنین لازم به ذکر است که در وکتور آدرس ها تغییر آدرس برای ما میسر نبوده.

بخش دوم مربوط به وقتی است که در حالی که یک وقفه در حال اجرا است وقفه ی دیگر سر می رسد که در این حالت ما فقط میتوانیم برای دو وقفه پیاده سازی کنیم.(خودمان میتوانیم برای وقفه ها اولویت در نظر بگیریم).

فرض کردیم مثلا اولویت وقفه مربوط به تایمر صفر بالاتر است و تایمر یک اولویت پایینتر دارد ینی زمانی که وقفه تایمر صفر در حال اجرا است اگر تایمر یک برسد نباید سراغ تایمر یک برود چون اولویت پایینتر دارد.پس در بدنه تابع مربوط به وقفه تایمر صفر نباید بیت INTERRUPTرا فعال کنیم ولی در بدنه تابع وقفه تایمر یک SEIرا ست کرده و فلگ INTERRUPTرا فعال میکنیم تا اگر وقفه تایمر صفر رسید بتواند آن را با توجه به اولویت بالاترش اجرا کند.

اما با این روش نمیتوان بیشتر از دو وقفه را هندل کرد.چون در این صورت برای وقفه های وسطی توصیف مشخصی از شرایطشان وجود ندارد و نه میتوان بیت INTERRUPTرا فعال کرد و نه میتوان غیر فعال کرد و قطعا در هر دو حالت شرایطی به وجود می آید که وقفه ها به درستی اجرا نشوند.

پس برای بخش دوم درحالتی که دو وقفه موجود است پیاده سازی انجام دادیم.