پیش‌گزارش دستورکار دوم آزمایشگاه ریزپردازنده و زبان اسمبلی

نگار موقتیان، ۹۸۳۱۰۶۲

۱. کدهای مورد نیاز برای برنامه ریزی برد:

فایل مربوط به کدهای نوشته شدهٔ برنامه به پیوست ارسال می‌گردد.

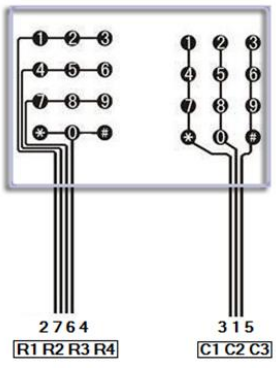
ابتدا تنظیمات مربوطه به keypad انجام شده‌اند (در این آزمایش از یک keypad 4x4 استفاده شده است). در تابع setup سریال راه اندازه شده و پین‌های خروجی برد مشخص شده‌اند. به علاوه برای سهولت درگرفتن ورودی از virtual terminal میزان timeout آن به ۲۵۰ میلی ثانیه کاهش داده شده است.

تابع setLED برای روشن کردن LED ها مطابق خواستهٔ دستور کار نوشته شده است. این تابع به عنوان ورودی عددی دریافت می‌کند که نشانگر تعداد LED هایی که باید روشن شوند است. ابتدا این عدد بررسی می‌شود تا مطمئن شویم عددی بین ۰ تا ۹ است و در غیر این صورت پیغام خطای مناسب را چاپ کنیم. سپس به ترتیب از سمت چپ LED هایی که باید روشن باشند را روشن کرده و باقی را خاموش می‌کند.

در تابع loop نیز بررسی می‌کنیم که عددی در keypad و یا virtual terminal وارد شده است یا خیر. در صورتی که عددی وارد شده بود ابتدا مقدار آن در ترمینال چاپ شده و سپس تابع setLED صدا زده می‌شود (عدد ورودی به عنوان آرگومان به این تابع پاس داده می‌شود).

۲. انواع keypad ماتریسی و چگونگی کارکرد آن‌ها:

یک نوع ساده از این keypad ها keypad های عادی هستند که صفحهٔ آن‌ها از تعدادی میکروسوئیچ به ازای هر دکمه تشکیل شده. با فشار دادن هر دکمه و اتصال میکروسوئیچ متناظر با آن می‌توان فشرده شدن دکمه را تشخیص داد. در این keypad های ماتریسی هر سطر و هر ستون یک پین متناظر دارد. پایه‌های R1 تا R4 به پایه‌های خروجی میکروکنترلر متصل می‌شوند و در حالت عادی ولتاژ HIGH دارند. پایه‌های C1 تا C3 نیز به صورت ورودی و در حالت pull-up در نظر گرفته می‌شوند. به صورت نرم‌افزاری و در توابع از پیش تعریف شده در



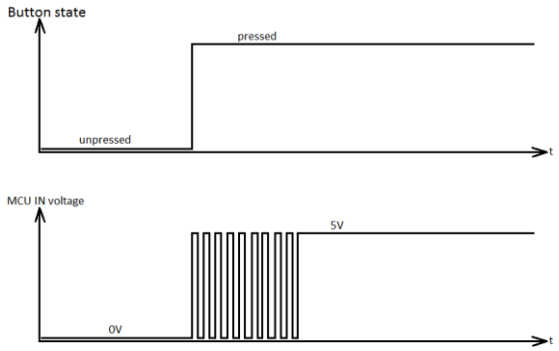
شکل ۱ – نمونه‌ای از یک keypad ماتریسی عادی

کتابخانهٔ Keypad مقدار پایه‌های R به ترتیب LOW می‌شوند و به ازای هر یک بررسی می‌شود که آیا هیچ یک از پایه‌های C نیز LOW شدند یا خیر. اگر برای مثال پایهٔ R_i را LOW کرده بودیم و دیدیم که پایهٔ C_j هم LOW شده است، این بدین معناست که دکمه موجود در ردیف i و ستون j فشرده شده‌است (با این روش فشرده شدن همزمان چند دکمه نیز قابل تشخیص است).

نوع دیگر keypad ها keypad های خازنی هستند که در آن‌ها صفحات به خازن‌های شارژ شده‌ای متصل هستند. با تماس دست با آن‌ها این خازن‌ها تخلیه شده و سیستمی که در این keypad ها تعبیه شده تشخیص می‌دهد که کدام دکمه لمس شده است.

۳. پدیدهٔ نوسان (bounce) کلید چیست و چگونه می توان از بروز اشکالات ناشی از آن جلوگیری کرد؟

فرض کنید یکی از پایه های ورودی به یک دکمه فشاری متصل است (یا یک دکمه از keypad). ما یک بار دکمه را فشار می دهیم اما در واقعیت در خروجی این دکمه یک انتقال تمیز از صفر به یک نداریم. به دلیل این که کلید فیزیکی است ممکن است چندین بار اتصال قطع و وصل شود تا بالاخره ثابت بماند. این قضیه باعث می‌شود پردازنده به جای یک بار فشرده شدن دکمه چندین تغییر را حس کند و برنامه کارکرد مورد انتظار را نداشته باشد. برای حل این مشکل می‌توان از debouncing استفاده کرد. Debouncing باعث می شود تغییر دکمه تنها یکبار به پردازنده خبر داده شود (این کار با بررسی عرض پالس انجام می‌شود). در میکروکنترلری که با آن کار می‌کنیم می‌توان این قابلیت را بر روی پایهٔ ورودی فعال کرد تا از بروز این مشکل جلوگیری کنیم. به علاوه می‌توان از یک خازن یا یک latch برای جلوگیری از این پدیده استفاده کرد.



شکل ۲ – سیگنالی به شکل پایین می‌تواند توسط debouncing به سیگنال بالا تبدیل شود.

۴. تعریف مختصر توابع مورد نیاز از کتابخانه Keypad.h:

- Keypad(makeKeymap(userKeymap), row[], col[], rows, cols)

این تابع در حقیقت constructor کلاس Keypad برای ایجاد یک شیء جدید از این کلاس می‌باشد. آرگومان اول آن آرایهٔ کاراکترهای موجود بر روی keypad می‌باشد که خود به تابع makeKeymap پاس داده شده تا به فرمت مناسب تبدیل شود. آرگومان بعدی آن آرایه‌ای از شماره پین‌های متصل شده به سطرهای keypad و آرگومان بعدی نیز آرایه‌ای از شماره پین‌های متصل شده به ستون‌های keypad می‌باشد. پس از آن نیز باید تعداد سطرها و ستون‌های keypad را مشخص نماییم.

- char getKey()

دکمه‌ای که فشرده شده است را در صورت وجود برمی‌گرداند. این تابع non-blocking است، بدین معنا که تنها بررسی می‌کند که دکمه‌ای فشرده شده یا خیر و منتظر فشرده شدن دکمه‌ای نمی‌شود (بدین ترتیب ادامهٔ کد می‌تواند در همان لحظه اجرا شود و اجرای برنامه متوقف نمی‌شود).

- bool getKeys()

مشخص می‌کند که آیا تعدادی دکمه فشرده شده‌اند یا خیر. این تابع تنها یک مقدار boolean بر می‌گرداند و برای مشخص کردن این که کدام دکمه‌ها فشرده شده‌اند باید از تابع دیگری استفاده کنیم.

- char waitForKey()

مانند getKey مقدار دکمهٔ فشرده شده را برمی‌گرداند، با این تفاوت که blocking است تا ابد منتظر می‌ماند تا کاربر دکمه‌ای را فشار دهد. زمانی که این تابع منتظر ورودی کاربر است هیچ یک از کدهای موجود پس از این تابع اجرا نمی‌شوند.

- KeyState getState()

حالت هر یک از دکمه‌ها را برمی‌گرداند. این حالات می‌توانند یکی از مقادیر IDLE (کلید فشرده نشده)، PRESSED (کلید در این لحظه فشرده شده)، RELEASED (کلید در این لحظه رها شده) و HOLD (کلید فشرده نگه داشته شده - معیار فشرده نگه داشته شدن کلید با استفاده از تابع setHoldTime مشخص می‌شود) را به خود بگیرند.

- boolean keyStateChanged()

اگر حالت هر یک از دکمه‌های Keypad تغییر کند می‌توانیم این تغییر را با استفاده از خروجی این تابع متوجه شویم.

۵. نحوه و کاربردهای ارتباط سریال در آردوینو:

ارتباط سریال برای ارتباط میان دو دستگاه مانند دو میکروکنترلر یا یک میکروکنترلر و کامپیوتر از طریق یک پروتکل مشخص استفاده می‌شود. برای یک ارتباط دو طرفهٔ سریال حداقل به ۳ سیم نیاز داریم. یک سیم GND که ولتاژ مرجع میان دو برد را تعیین می‌کند و دو سیم دیگر به نام سیم‌های RX و TX که یکی مربوط به خروجی و دیگری مربوط به ورودی می‌شود (R از کلمهٔ Receive و T از کلمهٔ Transfer می‌آید). در واقع دو دستگاهی که از طریق ارتباط سریال به یکدیگر متصل می‌شوند از طریق یکی از این سیم‌ها داده را ارسال کرده و از طریق دیگری داده را دریافت می‌کنند، لذا پین RX یک برد باید به TX دیگری و پین TX آن باید به RX دیگری متصل شود.

برد آردوینویی که از آن استفاده می‌کنیم از ۴ ارتباط سریال متناظر با پایه‌های TX0 و RX0 (که پایه‌های ۰ و ۱ برد هستند) تا TX3 و RX3 پشتیبانی می‌کند. در برنامه‌ای که می‌نویسیم این سریال‌ها با Serial، Serial1، Sertial2 و Serial3 تعریف شده‌اند. علاوه بر این پایه‌ها می‌توان از دیگر پین‌ها برای ارتباط سریال استفاده کرد اما در این صورت مدیریت آن‌ها به صورت نرم‌افزاری و توسط کتابخانهٔ SoftwareSerial انجام می‌شود. هر یک از این Serial ها می‌تواند داده‌ای را بخواند یا بنویسد.

۶. تعریف مختصر و نحوه کار با توابع ارتباط سریال:

- ❖ begin()

این تابع اتصال سریال را برقرار می‌کند و از آن لحظه سریال آمادهٔ ارسال و دریافت اطلاعات می‌شود. این تابع به عنوان آرگومان سرعت ارتباط سریال را بر حسب بیت بر ثانیه دریافت می‌کند. همچنین یک آرگومان optional به نام config دارد که برای یکسری از تنظیمات به کار می‌رود (برای مثال برای تنظیم بیت parity و ...).

- ❖ end()

بر عکس تابع قبل ارتباط سریال را می‌بندد و پین‌های RX و TX را برای دیگر استفاده‌ها آزاد می‌کند.

- ❖ find()

داده‌ها را از بافر سریال می‌خواند تا زمانی که به داده‌ای مشخص (که به عنوان آرگومان به این تابع داده می‌شود) برسد. در صورت وجود چنین داده‌ای true و در غیر این صورت false برمی‌گرداند.

- parseInt()

به دنبال اولین عدد معتبر در ورودی سریال می‌گردد. اگر در زمان مشخصی (پیش از زمان timeout) چنین عددی را دریافت کرد آن را برگردانده و در غیر این صورت مقدار صفر برمی‌گرداند. به علاوه آرگومان‌هایی optional دارد که یکی برای مشخص کردن عملکرد برنامه زمانی که ورودی مخلوطی از اعداد و دیگر کاراکترها باشد و یکی برای کاراکترهایی که می‌توانند زمان parse کردن نادیده گرفته شوند به کار می‌رود.

- println()

داده‌ای را به صورت متن قابل فهم برای انسان به خروجی سریال می‌فرستد (با استفاده از کاراکترهای ASCII)، در انتها نیز به خط بعد می‌رود. به علاوه می‌توان یک فرمت نیز برای آن تعیین کرد، برای مثال مشخص کنیم که خروجی به فرمت باینری باشد، یا هگزادسیمال یا... .

- read()

اولین بایت از ورودی سریال را خوانده و آن را برمی‌گرداند.

- readStringUntil()

ورودی سریال را تا زمان دریافت یک کاراکتر خاص (که به عنوان ورودی به آن داده می‌شود) و یا سپری شدن timeout دریافت کرده و رشته دریافت شده را به صورت یک String برمی‌گرداند.

- write()

داده‌ای را به صورت رشته بیت در خروجی سریال می‌نویسد. به عنوان ورودی می‌تواند یک مقدار خاص، یک String و یا یک آرایه به صورت بافر (به همراه طول آن) را دریافت کرده، آن را به یک یا تعدادی بایت تبدیل کرده و در خروجی بنویسد.