Хардуер планиране

// 6н2075в избрах защото е в наличност, има достатъчен „коефициент на усилване“

// R1 е избран така, че да ограничава ток с размер 2 мА. 2мА е избрана стойност, защото няма да изгори GPIO пин-а, но е достатъчна за да задейства оптрона. Когато оптронът се отпуши, той пропуска 12 волта напрежение, което отпуска дарлингтон транзистора, който вече увеличава тока до нужните стойности за ключалката (тя има нужда от 430 мА). Също така, тази схема е по-евтина от подобна такава с електронен елемент реле.R2 и R3 са резистори, чиято цел е да ограничават тока, за да не изгори системата. D1 е стандартен диод, който пази транзистора Q1 повреда, поради индукцията, предизвикана от бобината в ключалката. Той позволява на тока да се разсея и елиминира потенциален скок в напрежението. 12 волта идват от контакт.

//

Pn532 сензора може да бъде захранван от 5 V от RPI, защото нормалната консумация е около 150 мА, а РПИ има товароспособност на двата 5В пина общо от около 300 мА. Те са свързани, за да осигурят достатъчен ток при евентуални пикови ситуации от страна на сензора. GRND също е свързан с този на Пито

// Трите ЛЕД-а служат за индикация Р4 Р5 и Р6 ще бъдат пресметнати спрямо диодите и тяхното светене, когато ги набавя, за да преценя колко силна да е светлината.

// Използвам SN74HC04N чип, за да контролирам LED сигналите през GPIO безопасно. Чипът позволява контролирането на диодите с голям ток (4мА на диод), управляван от малък ток на входа на канала на инвертора (2мА). Този елемент е избран, поради гъвкавото си захранване (между 2 и 6 волта май) и ниската си цена (50ст). Позволява слагането на още 3 диода при нужда за разширение или монтиране на втори такъв панел от другата страна на вратата.

//

IOManager – разделен на .hpp и .cpp притежава атрибути: stream – file stream ot std, string filename – името на file-а с който ще работи, pthread mutex, който заключва писането и четенето, за да позволи thread-safe работа с файловете. Write получава като аргумент данни, които записва във файл след като заключи мютекс. Read чете от файл, като записва в string, подаден като аргумент, прави го след заключване на мютекс. Конструктора инициализира мютекса, а деструктора го унищожава. Копи конструктор и move? конструктора се генерират автоматично от компилатора. Избрах posix threads, защото са поддържани на много платформи (android, linux, Windows with added custom library, etc..). Много по-стабилни и основни са от std::thread (който е построен чрез posix threads). Всичко е документирано както с Doxygen, така и с ръчно написани коментари.

Причината да избера тези gpio-та е, че при буутване състоянието по подразбиране е input с pull-down резистори (0), което означава, че при стартиране няма да доведе до отключване на бравата докато се зареди програмата. (GPIOs 0 to 8 Pull-up, другите са pull-down). Причината да инициализирам GPIO-тата като входни (инпут) е, че когато е в режим на вход, входното съпротивление е много високо и това би спасило веригата от евентуално късо съединение при грешно свързване.

Какъв е хардуера? Как е избран? Какво притежава? (Pi-to, pn532 .. etc)

Източници

<https://www.gnu.org/software/libc/manual>

книги

1. Защо джанго
2. Инсталация на virtualenv??
3. Инсталация на джанго + документацията му
4. Създаване на проект
5. Обясн ение на различните файлове?
6. Създаване на апликация
7. Създаване на пътищата
8. View-ta
9. Какво и защо html
10. Създаване на template (podavane na context) {% statement %} {% endstatement%} {{ variable }} inheritance
11. Include CSS
12. Mysqlclient python library
13. Create Users
14. Read Users
15. Update User
16. Delete User