Если вы видите что-то необычное, просто сообщите мне.

Raspberry Pi и программирование на Go

Go или GoLang - компилируемый язык программирования разработанный Google.

Go широко используется для web разработки и на сегодня это самый быстро растущий язык. В отличии от Java который запускается в JVM, Go собирается прямо в Windows, OS X or Linux исполняемые файлы.

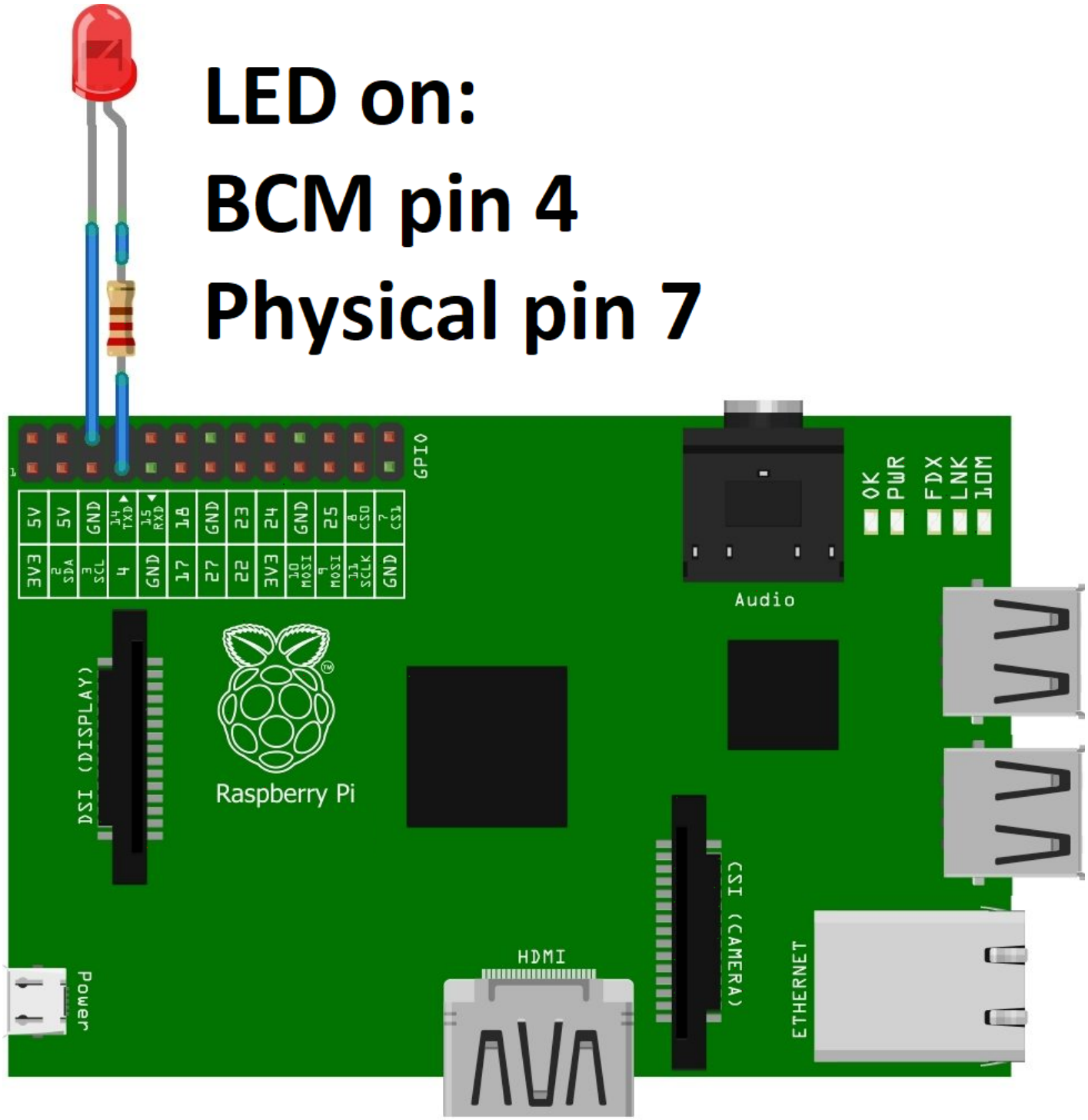
В этой статье мы взглянем на создание двух програм, которые разговаривают с Raspberry Pi GPIO. Первая будет просто клавиатурный ввод программы, а вторая будет отдельным web приложением управляющим пинами GPIO.

Raspberry Pi GPIO

Есть можество различных способов производить подключение к GPIO. Для нашего примера мы собираемся посмотреть на вывод gpio терминальной утилиты, так же будем использовать go-rpi библиотеку.

Для тестирования мне нравится использовать gpio утилиту так как она предлагает хороший выбор команд и я могу в ручном режиме протестировать и проверить команды, прежде, чем я буду использовать их в коде. Для помощи можно воспользоваться ключем `-h`.

Аппаратное обеспечение Raspberry PI настраивается используя резистор на физическом пине 7(BCM пин 4)



Наша первая программа(keyin.go) будет читать ввод клавиатуры и затем направлять в gpio дважды, первый раз, чтобы записать значение, второй раз чтобы прочитать значение обратно.

```
package main

import (
    "bufio"
    "fmt"
```

```
    "os/exec"
    "os"
)

func main() {
    // Get keyboard input
    reader := bufio.NewReader(os.Stdin)
    fmt.Print("Enter value for GPIO pin 7 : ")
    pinval, _ := reader.ReadString('\n')

    // Write to GPIO pin 7 using keyboard input
    testCmd := exec.Command("gpio","write", "7", pinval)
    testOut, err := testCmd.Output()
    if err != nil {
        println(err)
    }
    // Read back the GPIO pin 7 status
    testCmd = exec.Command("gpio","read", "7")
    testOut, err = testCmd.Output()
    if err != nil {
        println(err)
    } else {
      fmt.Print("GPIO Pin 4 value : ")
      fmt.Println(string(testOut))
    }
}
```

Скомпилируем и запустим keyin.go программу:

```
 $ go build keyin.go
 $ ./keyin
Enter value for GPIO pin 7 : 1
GPIO Pin 4 value : 1
```

Простое Go веб-приложение

Для начального примера мы сделаем веб-приложение(web_static.go) которое показывает страничку `web_static.html`.

web_static.html выглядит следующим образом:

```
<html>
  <head>
    <title>GO PI Static Page</title>
  </head>
  <body>
    <h1>GO PI Static Page</h1>
    <hr>
    This is a static test page
  </body>
</html>
```

`web_static.go` программе необходимо импортировать `net/http` библиотеку. `http.HandleFunc` вызов используется для стандартного адреса “/” и раздачи с помощью сервера нашего `web_static.html` файла. `http.ListenAndServe` фукнция слушает web запросы на порту 8081.

```
package main

import (
    "log"
    "net/http"
)

func main() {
    // Create default web handler, and call a starting web page
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        http.ServeFile(w, r, "web_static.html")
        println("Default Web Page")
```

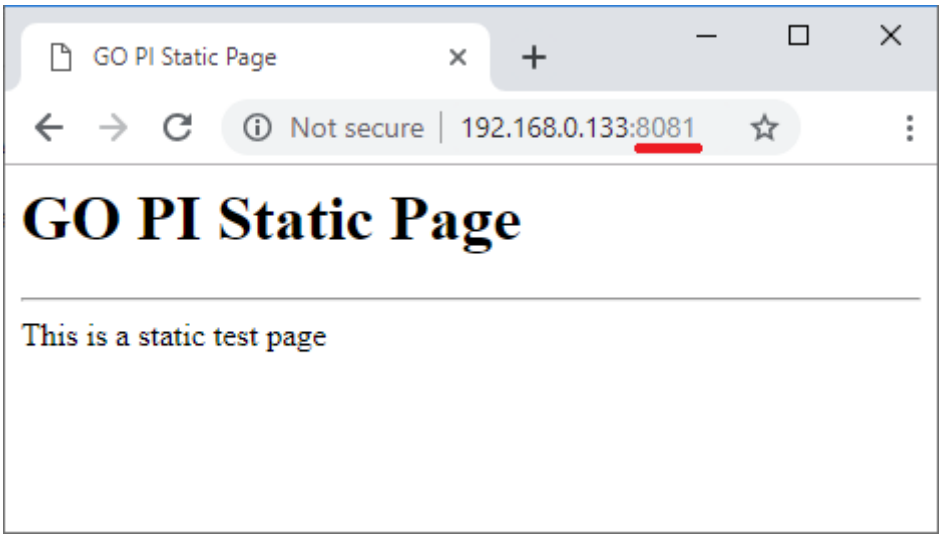
```
    })
    // start a listening on port 8081
    log.Fatal(http.ListenAndServe("8081", nil))

}
```

Go код может быть скомпилирован и запущен.

```
$ go build web_static.go
$ ./web_static
Default Web Page
```

Из web бразуера указывающего на Raspberry Pi и порт 8081,наше приложение будет выглядит так:



Go Web приложение с Pi GPIO

Следующий шаг - создание web страницы которая может передавать параметры. Для этогоп риложения мы переключим GPIO вывод с `on` на `off`.

Новая web страница(go_buttons.html) создается с двумя кнопками. Html якорь используется для передачи `/on` и `/off` параметров в наше web приложение.

`CACHE-CONTROL` тег устанавливается в `NO-CACHE` для того, чтобы мы точно знали, что страница перезагружается. Я так же включил `EXPIRES` тег равным нулю, чтобы браузер всегда видел страничку "испорченной". Если вы не включаете их то страница может быт загружена только один раз.

```
<html>
  <head>
    <title>GO PI GPIO</title>
    <META HTTP-EQUIV="CACHE-CONTROL" CONTENT="NO-CACHE">
    <META HTTP-EQUIV="EXPIRES" CONTENT="0">
  </head>
  <body>
    <h1>Go Raspberry Pi GPIO Test</h1>
    <a href="/on"><button>Set LED ON</button></a><br>
    <a href="/off"><button>Set LED OFF</button></a>
  </body>
</html>
```

Наше новое web приложение(go_buttons.go) теперь включает еще две `http.HandleFunc` функции обработчика, одну для `/on` и одну для `/off`. Эти обработчики вызывают функцию которая вызывает gpio, что используется для написания вывода и чтения обратно статуса ввода.

Наша новоиспеченная gpio функция выполняет ходит 2 раза к gpio командной утилите, первый раз записать значение, второй - прочитать значение обратно.

```
package main

import (
"log"
"net/http"
"os/exec"
)
func gpio( pinval string) {
    testCmd := exec.Command("gpio","write", "7", pinval)
    testOut, err := testCmd.Output()
    if err != nil {
        println(err)
    }
```

```
    testCmd = exec.Command("gpio","read", "7")
    testOut, err = testCmd.Output()
    if err != nil {
        println(err)
    } else {
      print("GPIO Pin 4 value : ")
      println(string(testOut))
    }
}

func main() {

	http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
		http.ServeFile(w, r, "go_buttons.html")
		println("Default Web Page")
	})

	http.HandleFunc("/on", func(w http.ResponseWriter, r *http.Request) {
                http.ServeFile(w, r, "go_buttons.html")
		println("Web Page sent : ON")
		gpio("1")

	})

	http.HandleFunc("/off", func(w http.ResponseWriter, r *http.Request) {
                http.ServeFile(w, r, "go_buttons.html")
		println("Web Page sent : OFF")
		gpio("0")
	})

	log.Fatal(http.ListenAndServe(":8081", nil))

}
```

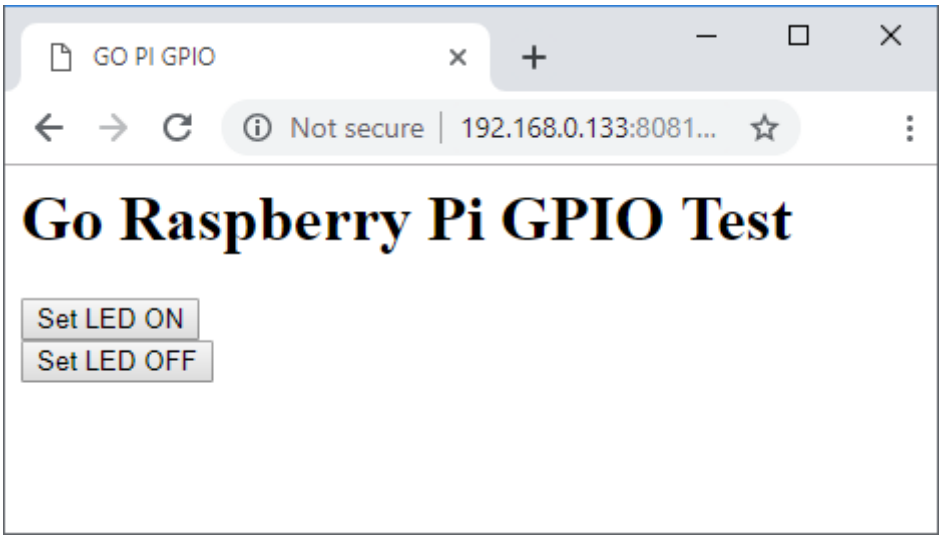
Скомпилируем и запустим go_buttons:

```
$ go build go_buttons.go
$ ./go_buttons
```

```
Default Web Page
Web Page sent : ON
GPIO Pin 4 value : 1

Default Web Page
Web Page sent : OFF
GPIO Pin 4 value : 0
```

The web page should look something like:



Выводы

Для конечного приложения я бы предпочёл использовать Go нативную библиотеку для вызовов GPIO, но для прототипирования я нашел, что командная утилита проще в решении проблем.

Revision #3
Created 2021-10-22 12:14:11 UTC by gasick
Updated 2023-04-16 19:30:04 UTC by gasick