Woolsey Workshop

Cheatsheet: Raspberry Pi Basics

General

Resources

- Raspberry Pi Foundation website
- Raspberry Pi Products (contains Getting Started, Specifications, and general documentation)
- Raspberry Pi Help (various guides and documentation)
- Raspberry Pi SSH (Secure Shell)
- Raspberry Pi Raspbian
- Raspberry Pi Hardware details
- Raspberry Pi GPIO Pin-Out (useful in understanding GPIO functionality and numbering schemes)
- Woolsey Workshop Raspberry Pi Tutorials

General Information

- Raspbian is the officially recommended OS for Raspberry Pi systems. It is free, based on the standard Debian Linux distribution, and is optimized for Raspberry Pi hardware.
- The default username is *pi* and password is *raspberry*.
- The Raspberry Pi Configuration application allows you to enable or change system wide configurations, such as GPIO interfaces, Secure Shell, etc.
- Always verify which pin numbering scheme is being used within instructions or tutorials so that you not only connect your circuits properly, but that you don't accidentally damage your Raspberry Pi. The following three schemes are typically used: Broadcom, Physical, and WiringPi.
- Whether you are working with the GPIO pins from the command line or from within programs, it is good practice to reset the pins back to an input (their default state) in order to avoid accidental damage to the pins or interfere with other programs.

Operation

- Open Raspberry Pi configuration tool
  - GUI: Raspberry Menu > Preferences > Raspberry Pi Configuration
  - CLI: `% sudo raspi-config`
- Open terminal window from GUI
  Click on LXTerminal application (icon with >_ in menu bar)
- Access privileged commands (when permission is denied)
  `% sudo <command>`
- Search for new package
  `% apt-cache search <package> | sort`
- Install new package
  `% sudo apt-get install <package>`
- Upgrade installed packages
  ```
  % sudo apt-get update
  % sudo apt-get upgrade
  ```

- Upgrade entire operating system
  ```
  % sudo apt-get update
  % sudo apt-get dist-upgrade
  % sudo reboot
  ```
- Enable on-board interfaces
  - Raspberry Pi Configuration (GUI)
    Open Raspberry Pi configuration tool (Raspberry Menu > Preferences > Raspberry Pi Configuration).
    Go to the Interfaces Tab and select Enable for the interfaces you want to enable, then click Ok and reboot if prompted.
  - Raspi-Config (CLI)
    ```
    % sudo raspi-config
    ```
    Select Interfacing Options, then the interface you want to enable, then Yes, then Ok, then Finish and reboot if prompted.
- Display I2C addresses in use
  ```
  % i2cdetect -y 1
  ```
- Show pinout diagram
  - Website: http://pinout.xyz
  - Raspbian CLI: `% pinout`
  - WiringPi CLI: `% gpio readall`

Start-Up

1. Complete all circuit wiring.
2. Connect the Raspberry Pi to power.

Shutdown

1. Save all current work.
2. Shutdown Raspbian OS
   - GUI: Raspberry Menu > Shutdown…
   - CLI: `% sudo shutdown -h now`
3. Disconnect the Raspberry Pi from power.

Power Pins

- 5 V rated at ? mA (current is limited by power usage of all active interfaces across the entire board, but at least 100 mA seems to be the prevailing answer in the forums)
- 3.3 V rated at 50 mA

Digital I/O Pins

- Rated 3.3 V @ 16 mA (maximum of 50 mA across all pins simultaneously)
- All GPIO pins revert to general-purpose inputs on power-on reset.
- Any pin can be configured as an input or output.
- Any pin can be configured as an interrupt.

- Pins *GPIO2* and *GPIO3* have fixed pull-up resistors. All other pins can be configured as pull-up or pull-down.
- Hardware PWM is available on *GPIO12*, *GPIO13*, *GPIO18*, and *GPIO19*. Software PWM is available on all GPIO pins.
- If more current is required on an output, a transistor can be used to drive the component powered by an external power supply.

Pin-Out

1. **3V3** - 3.3 V Power Supply
2. **5V** - 5 V Power Supply
3. **GPIO2/SDA** - Digital I/O / I2C SDA
4. **5V** - 5 V Power Supply
5. **GPIO3/SCL** - Digital I/O / I2C SCL
6. **GND** - Ground
7. **GPIO4** - Digital I/O
8. **GPIO14/TX** - Digital I/O / UART TX
9. **GND** - Ground
10. **GPIO15/RX** - Digital I/O / UART RX
11. **GPIO17/CE1** - Digital I/O / SPI1 CE1
12. **GPIO18/CE0/PWM0** - Digital I/O / SPI1 CE0 / Pulse Width Modulation
13. **GPIO27** - Digital I/O
14. **GND** - Ground
15. **GPIO22** - Digital I/O
16. **GPIO23** - Digital I/O
17. **3V3** - 3.3 V Power Supply
18. **GPIO24** - Digital I/O
19. **GPIO10/MOSI** - Digital I/O / SPI0 MOSI
20. **GND** - Ground
21. **GPIO9/MISO** - Digital I/O / SPI0 MISO
22. **GPIO25** - Digital I/O
23. **GPIO11/SCK** - Digital I/O / SPI0 SCK
24. **GPIO8/CE0** - Digital I/O / SPI0 CE0
25. **GND** - Ground
26. **GPIO7/CE1** - Digital I/O / SPI0 CE1
27. **GPIO0** - Reserved (EEPROM Communication)
28. **GPIO1** - Reserved (EEPROM Communication)
29. **GPIO5** - Digital I/O
30. **GND** - Ground
31. **GPIO6** - Digital I/O
32. **GPIO12/PWM0** - Digital I/O / Pulse Width Modulation
33. **GPIO13/PWM1** - Digital I/O / Pulse Width Modulation
34. **GND** - Ground
35. **GPIO19/MISO/PWM1** - Digital I/O / SPI1 MISO / Pulse Width Modulation
36. **GPIO16/CE2** - Digital I/O / SPI1 CE2
37. **GPIO26** - Digital I/O
38. **GPIO20/MOSI** - Digital I/O / SPI1 MOSI
39. **GND** - Ground
40. **GPIO21/SCK** - Digital I/O / SPI1 SCK

WiringPi GPIO Command Line Utility

- The Wiring Pi GPIO Utility can be utilized from the command line or within scripts to manipulate GPIO pins.
- Pins can be interpreted with an option flag as standard WiringPi (default), BCM (-g), or physical (-1) pin numbers.
- Install WiringPi
  ```
  % sudo apt-get install wiringpi
  ```
- View man page
  ```
  % man gpio
  ```
- Print version
  ```
  % gpio -v
  ```
- Print help
  ```
  % gpio -h
  ```
- Read all pins
  ```
  % gpio readall
  ```
- Set pin mode
  ```
  % gpio mode <pin> <mode>
  ```
  where <pin> = WiringPi pin number; <mode> = in, out, pwm, clock, up, down, tri
- Read pin value
  ```
  % gpio read <pin>
  ```
  where <pin> = WiringPi pin number
- Set pin value
  ```
  % gpio write <pin> <value>
  ```
  where <pin> = WiringPi pin number; <value> = 0 or 1

Programming

- Many programming languages come preinstalled on Raspbian and many more can be installed.
- RPi.GPIO and GPIO Zero are popular Python based GPIO interface libraries.
- WiringPi is a popular C based GPIO interface library.
- SwiftyGPIO is a popular Swift based GPIO interface library.

Example Programs

Example Blink Program Using C And WiringPi

Compile: `% gcc -Wall -lwiringPi -o blink blink.c`
Run: `% ./blink`
Exit: CTRL-C

```
// blink.c
//
// Description:
// C program to blink an LED on a Raspberry Pi with the WiringPi library.
//
// Created by John Woolsey on 05/30/2018.
// Copyright © 2018 Woolsey Workshop.  All rights reserved.


#include <signal.h>
#include <stdio.h>
#include <wiringPi.h>


// Pin Definitions
const int redLED = 29;  // WiringPi pin 29 (BCM pin 21, physical pin 40)


// Global Variables
volatile sig_atomic_t signal_received = 0;  // signal interrupt received


// Signal Interrupt Handler
void sigint_handler(int signal) {
   signal_received = signal;  // capture interrupt signal
}


// Main
int main(void) {
   // Detect when CRTL-C is pressed
   signal(SIGINT, sigint_handler);  // enable interrupt handler

   // Initialize WiringPi library
   // (program will terminate if an error is encountered)
   wiringPiSetup();
```

```
   // Pin setup
   pinMode(redLED, OUTPUT);  // set redLED pin as an output

   // Blink LED
   printf("Press CTRL-C to exit.\n");
   while (!signal_received) {      // runs until CTRL-C is pressed
      digitalWrite(redLED, HIGH);  // turn on LED
      delay(500);                  // wait 500 milliseconds
      digitalWrite(redLED, LOW);   // turn off LED
      delay(500);                  // wait 500 milliseconds
   }

   // Exit cleanly when CTRL+C is pressed
   pinMode(redLED, INPUT);  // reset redLED pin as an input
   printf("\nCompleted cleanup of GPIO resources.\n");
   return(signal_received);
}
```

Example Blink Program Using C++ And WiringPi

Compile: `% g++ -Wall -lwiringPi -o blink blink.cpp`
Run: `% ./blink`
Exit: CTRL-C

```
// blink.cpp
//
// Description:
// C++ program to blink an LED on a Raspberry Pi with the WiringPi library.
//
// Created by John Woolsey on 05/30/2018.
// Copyright © 2018 Woolsey Workshop.  All rights reserved.


#include <csignal>
#include <iostream>
#include <wiringPi.h>


// Namespaces
using namespace std;


// Pin Definitions
const int redLED = 29;  // WiringPi pin 29 (BCM pin 21, physical pin 40)


// Global Variables
volatile sig_atomic_t signal_received = 0;  // signal interrupt received


// Signal Interrupt Handler
void sigint_handler(int signal) {
   signal_received = signal;  // capture interrupt signal
}


// Main
int main() {
   // Detect when CRTL-C is pressed
   signal(SIGINT, sigint_handler);  // enable interrupt handler

   // Initialize WiringPi library
   // (program will terminate if an error is encountered)
   wiringPiSetup();
```

```
   // Pin setup
   pinMode(redLED, OUTPUT);  // set redLED pin as an output

   // Blink LED
   cout << "Press CTRL-C to exit." << endl;
   while (!signal_received) {      // runs until CTRL-C is pressed
      digitalWrite(redLED, HIGH);  // turn on LED
      delay(500);                  // wait 500 milliseconds
      digitalWrite(redLED, LOW);   // turn off LED
      delay(500);                  // wait 500 milliseconds
   }

   // Exit cleanly when CTRL+C is pressed
   pinMode(redLED, INPUT);  // reset redLED pin as an input
   cout << endl << "Completed cleanup of GPIO resources." << endl;
   return(signal_received);
}
```

Example Blink Program Using Python And RPi.GPIO

Run: `% python blink_rpigpio.py`
Exit: CTRL-C

```
# blink_rpigpio.py
#
# Description:
# Python program to blink an LED on a Raspberry Pi with the RPi.GPIO
library.
#
# Created by John Woolsey on 05/30/2018.
# Copyright c 2018 Woolsey Workshop.  All rights reserved.


import RPi.GPIO as GPIO
import time


# Pin Definitions
redLED = 21  # BCM pin 21, physical pin 40


# Pin Setup
GPIO.setmode(GPIO.BCM)        # use BCM pin numbering
GPIO.setup(redLED, GPIO.OUT)  # set redLED pin as an output


# Blink LED
print("Press CTRL-C to exit.")
try:
   while True:                          # runs forever
      GPIO.output(redLED, GPIO.HIGH)  # turn on LED
      time.sleep(0.5)                 # wait half a second
      GPIO.output(redLED, GPIO.LOW)   # turn off LED
      time.sleep(0.5)                 # wait half a second


# Cleanup
finally:           # exit cleanly when CTRL+C is pressed
   GPIO.cleanup()  # release all GPIO resources
   print("\nCompleted cleanup of GPIO resources.")
```

Example Blink Program Using Python And GPIO Zero

Run: `% python blink_gpiozero.py`
Exit: CTRL-C

```
# blink_gpiozero.py
#
# Description:
# Python program to blink an LED on a Raspberry Pi with the GPIO Zero
library.
#
# Created by John Woolsey on 06/07/2018.
# Copyright c 2018 Woolsey Workshop.  All rights reserved.


from gpiozero import LED
from time import sleep


# Pin Definitions
redLED = LED(21)  # BCM pin 21, physical pin 40


# Blink LED
print("Press CTRL-C to exit.")
try:
   while True:      # runs forever
      redLED.on()   # turn on LED
      sleep(0.5)    # wait half a second
      redLED.off()  # turn off LED
      sleep(0.5)    # wait half a second


# Cleanup
finally:           # exit cleanly when CTRL+C is pressed
   redLED.close()  # release redLED resource
   print("\nCompleted cleanup of GPIO resources.")
```

Example Blink Program Using Swift And SwiftyGPIO

Create project with Swift Package Manager:

```
% mkdir blink
% cd blink
% swift package init --type executable
```

Edit *Package.swift* file to match the listing shown below.

```
% swift package update
```

Edit *Sources/blink/main.swift* file to match the listing shown below.

Compile: `% swift build`

Run: `% swift run`

Exit: CTRL-C

Package.swift file:

```
// swift-tools-version:4.0
// The swift-tools-version declares the minimum version of Swift required to
build this package.

import PackageDescription

let package = Package(
    name: "blink",
    dependencies: [
        // Dependencies declare other packages that this package depends on.
        .package(url: "https://github.com/uraimo/SwiftyGPIO.git", from:
"1.1.0"),
    ],
    targets: [
        // Targets are the basic building blocks of a package. A target can
define a module or a test suite.
        // Targets can depend on other targets in this package, and on
products in packages which this package depends on.
        .target(
            name: "blink",
            dependencies: ["SwiftyGPIO"]),
    ]
)
```

main.swift file:

```
// main.swift
//
// Description:
// Swift program to blink an LED on a Raspberry Pi with the SwiftyGPIO
library.
//
// Created by John Woolsey on 06/15/2018.
// Copyright © 2018 Woolsey Workshop.  All rights reserved.


import Foundation
import SwiftyGPIO


// Global Variables
var signalReceived: sig_atomic_t = 0  // signal interrupt received


// Pin Setup
let gpio = SwiftyGPIO.GPIOs(for: .RaspberryPi3)  // initialize SwiftyGPIO
library for use with the Raspberry Pi
guard let redLED = gpio[GPIOName.P21] else {  // # BCM pin 21, physical pin
40
   fatalError("Could not initialize redLED.")
}
redLED.direction = .OUT  // set redLED pin as an output


// Signal Interrupt Handler
signal(SIGINT) { signal in
   signalReceived = signal  // capture interrupt signal
}


// Blink LED
print("Press CTRL-C to exit.")
while signalReceived == 0 {  // runs until CTRL-C is pressed
   redLED.value = 1          // turn on LED
   usleep(500000)            // wait 500,000 microseconds
   redLED.value = 0          // turn off LED
   usleep(500000)            // wait 500,000 microseconds
}

// Exit cleanly when CTRL+C is pressed
redLED.direction = .IN  // reset redLED pin as an input
print("\nCompleted cleanup of GPIO resources.")
exit(signalReceived)
```

Revisions

Version	Date	Modifications
1.2	February 7, 2019	Updated general wording and formatting for better clarification.
1.1	January 24, 2019	Updated Swift example blink program to support Swift 4.
1.0	September 13, 2018	Initial version.