Francisco Javier Pedraza González

Arquitectura de Microcontroladores

CIATEQ

GPIO usage with bash and raspberry pi2 on a Yocto Project

Introduction

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It’s capable of doing everything you’d expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games.

What’s more, the Raspberry Pi has the ability to interact with the outside world, and has been used in a wide array of digital maker projects, from music machines and parent detectors to weather stations and tweeting birdhouses with infra-red cameras. We want to see the Raspberry Pi being used by kids all over the world to learn to program and understand how computers work.

RASPBERRY PI FOUNDATION

The Raspberry Pi Foundation is a registered educational charity (registration number 1129409) based in the UK. Our Foundation’s goal is to advance the education of adults and children, particularly in the field of computers, computer science and related subjects. See our stories page for more information about the Foundation’s charitable work.

Characteristics of Raspberry Pi2.

The Raspberry Pi 2 Model B is the second generation Raspberry Pi. It replaced the original Raspberry Pi 1 Model B+ in February 2015. Compared to the Raspberry Pi 1 it has:

A 900MHz quad-core ARM Cortex-A7 CPU

1GB RAM

Like the (Pi 1) Model B+, it also has:

4 USB ports

40 GPIO pins

Full HDMI port

Ethernet port

Combined 3.5mm audio jack and composite video

Camera interface (CSI)

Display interface (DSI)

Micro SD card slot

VideoCore IV 3D graphics core

Because it has an ARMv7 processor, it can run the full range of ARM GNU/Linux distributions, including Snappy Ubuntu Core, as well as Microsoft Windows 10 (see the blog for more information).

The Raspberry Pi 2 has an identical form factor to the previous (Pi 1) Model B+ and has complete compatibility with Raspberry Pi 1.

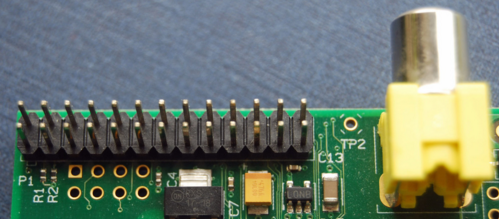
We recommend the Raspberry Pi 2 Model B for use in schools: it offers more flexibility for learners than the leaner (Pi 1) Model A+, which is more useful for embedded projects and projects which require very low power.

GPIO

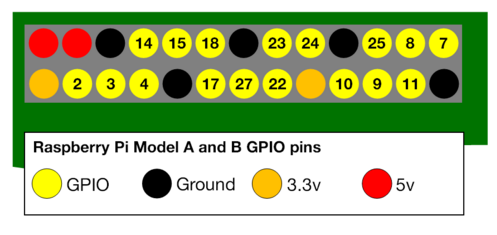
### AN INTRODUCTION TO GPIO AND PHYSICAL COMPUTING ON THE RASPBERRY PI

One powerful feature of the Raspberry Pi is the row of GPIO (general purpose input/output) pins along the edge of the board.

Circuit Diagram



These pins are a physical interface between the Pi and the outside world. At the simplest level, you can think of them as switches that you can turn on or off (input) or that the Pi can turn on or off (output). Seventeen of the 26 pins are GPIO pins; the others are power or ground pins.



## WHAT ARE THEY FOR? WHAT CAN I DO WITH THEM?

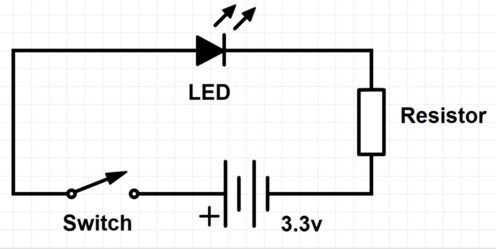
You can program the pins to interact in amazing ways with the real world. Inputs don't have to come from a physical switch; it could be input from a sensor or a signal from another computer or device, for example. The output can also do anything, from turning on an LED to sending a signal or data to another device. If the Raspberry Pi is on a network, you can control devices that are attached to it from anywhere\*\* and those devices can send data back. Connectivity and control of physical devices over the internet is a powerful and exciting thing, and the Raspberry Pi is ideal for this.

HOW THE GPIO PINS WORK

OUTPUT

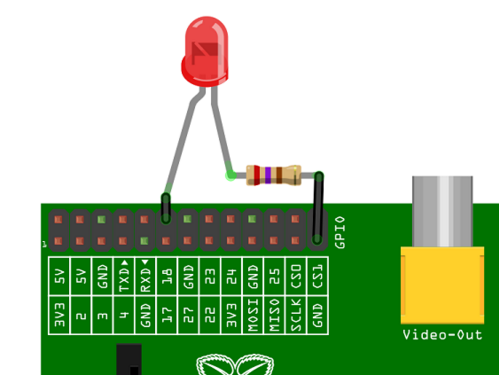
You have to be careful If you follow the instructions, then messing about with the GPIO is safe and fun. Randomly plugging wires and power sources into your Pi, however, may kill it. Bad things can also happen if you try to connect things to your Pi that use a lot of power; LEDs are fine, motors are not. If you are worried about this, then you might want to consider using a breakout board such as the Pibrella until you are confident enough to use the GPIO directly.

Ignoring the Pi for a moment, one of the simplest electrical circuits that you can build is a battery connected to a light source and a switch (the resistor is there to protect the LED):



When we use a GPIO pin as an output, the Raspberry Pi replaces **both the switch and the battery** in the above diagram. Each pin can turn on or off,or go HIGH or LOW in computing terms. When the pin is HIGH it outputs 3.3 volts (3v3); when the pin is LOW it is off.

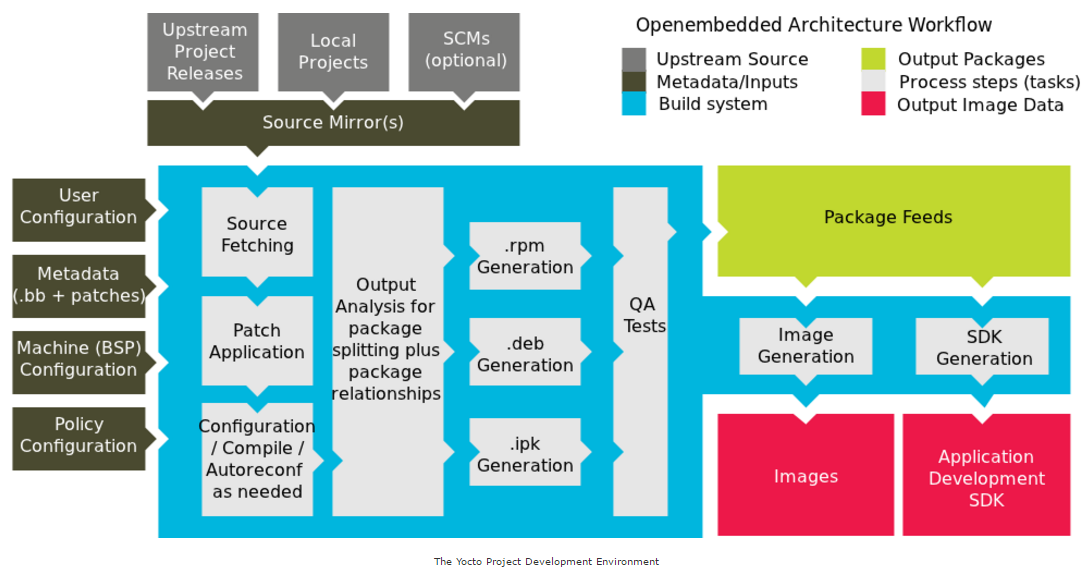
Here's the same circuit using the Raspberry Pi. The LED is connected to a GPIO pin (which can output +3v3) and a ground pin (which is 0v and acts like the negative terminal of the battery):



Configuring the OS

The Yocto Project is an open source collaboration project that provides templates, tools and methods to help you create custom Linux-based systems for embedded products regardless of the hardware architecture. It was founded in 2010 as a collaboration among many hardware manufacturers, open-source operating systems vendors, and electronics companies to bring some order to the chaos of embedded Linux development.

Development Environment



In order to setup the environment to compile your own Operative System you need to install some Features:

$ sudo dnf install gawk make wget tar bzip2 gzip python unzip perl patch \

diffutils diffstat git cpp gcc gcc-c++ glibc-devel texinfo chrpath \

ccache perl-Data-Dumper perl-Text-ParseWords perl-Thread-Queue socat \

This setup is for “Fedora” distribution, after this you need to configure your conf/local.conf and conf/bblayers.conf, also to clone the meta folders for the packages.

Clone yocto project and poky

$ git clone git://git.yoctoproject.org/poky

go to “poky” directory

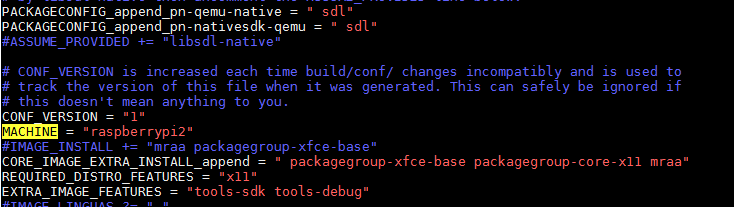
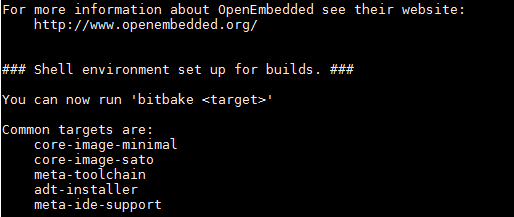
$ cd poky

Make source

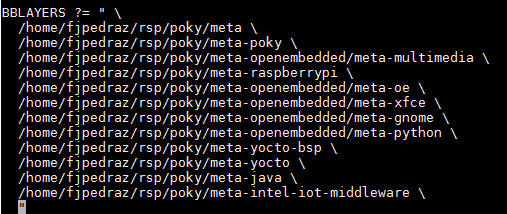
$ source oe-init-build-env

A default “build” folder should be created you can also create the folder name as you want in the third parameter of the command.

Then you have to modify the conf files



bblayers.conf



Compile the system

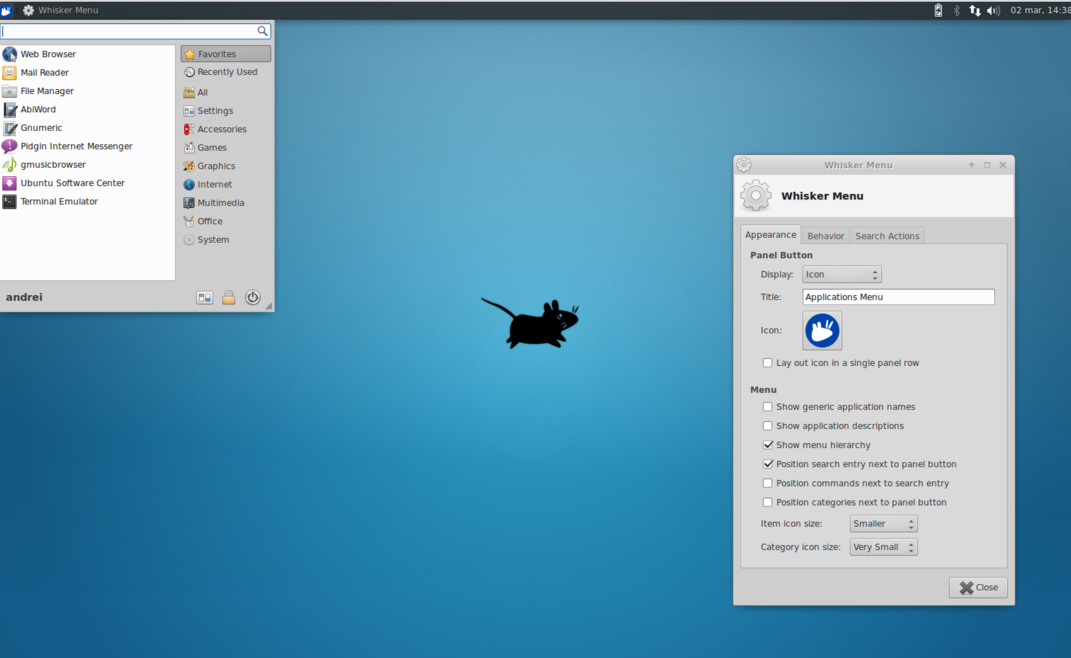
$ bitbake rpi-basic-image

It will generate an image with the previous configuration, then you have to burn a SD-Card, remember you have to use the device as “mmcblk0”.

Open a terminal and type”

$ sudo dd if=PATH\_to\_file.sdimg of=/dev/mmcblk0 bs=1M

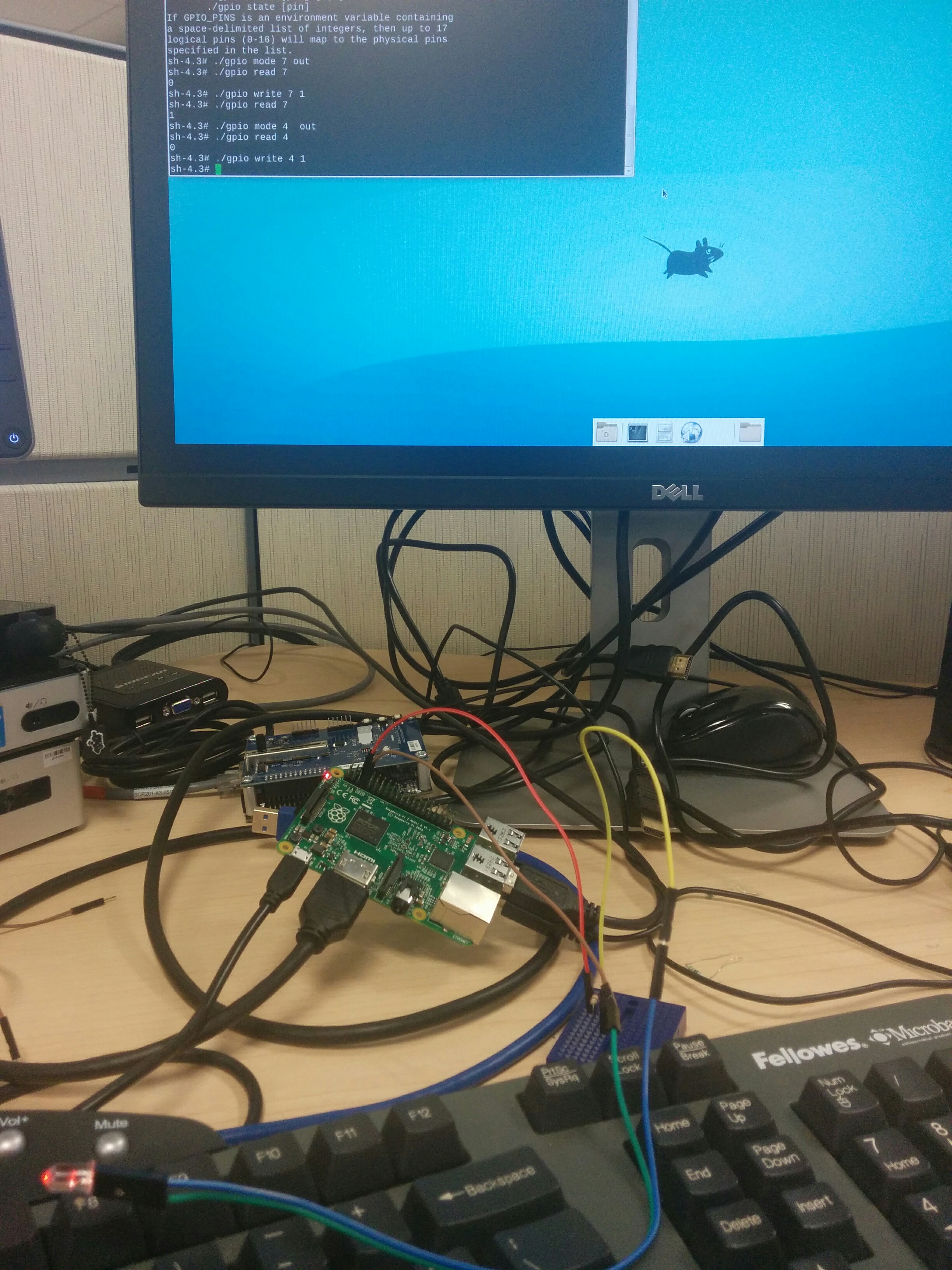
After this only you have to boot the burned image on the platform, and the generated Linux OS should be displayed successfully.



I have implemented the GPIO management with bash code.

Results

As an output two led are going to be displayed as a traffic light.



Code of the project

#!/bin/bash

# Binary LED counter for Raspberry Pi

GPIO\_PINS='0 1 4 7 8 9 10 11 14 15 17 18 21 22 23 24 25'

# Uncomment the following line for Raspberry Pi Revision 2

# GPIO\_PINS='2 3 4 7 8 9 10 11 14 15 17 18 22 23 24 25 27'

pins=17

delay=1

# Include gpio script to access the gpio function directly

source ./gpio

# Initialize pins

for ((pin = 0; pin < pins; pin++)); do

gpio mode $pin out

gpio write $pin 0

done

# Loop until CTRL-C is pressed

while true; do

for ((value = 0; value < 2 \*\* pins; value++)); do

remainder=$value

for((pin = pins - 1; pin >= 0; pin--)); do

power=$((2 \*\* pin))

digit=$((remainder / power))

remainder=$((remainder % power))

gpio write $pin $digit

done

sleep $delay

done

done

#Second File

#!/bin/bash

# LED chaser for Raspberry Pi

GPIO\_PINS='0 1 4 7 8 9 10 11 14 15 17 18 21 22 23 24 25'

# Uncomment the following line for Raspberry Pi Revision 2

#GPIO\_PINS='2 3 4 7 8 9 10 11 14 15 17 18 22 23 24 25 27'

pins=17

delay=0.025

# Include gpio script to access the gpio function directly

source ./gpio

# Initialize pins

for ((pin = 0; pin < pins; pin++)); do

gpio mode $pin out

done

pin=1

step=1

# Loop until CTRL-C is pressed

while true; do

gpio write $pin 1

sleep $delay

gpio write $pin 0

if ((pin == 0 || pin == pins - 1)); then

step=$((-step))

fi

pin=$((pin + step))

done

#Third File

#!/bin/bash

# Utility to control the GPIO pins of the Raspberry Pi

# Can be called as a script or sourced so that the gpio

# function can be called directly

function gpio()

{

local verb=$1

local pin=$2

local value=$3

local pins=($GPIO\_PINS)

if [[ "$pin" -lt ${#pins[@]} ]]; then

local pin=${pins[$pin]}

fi

local gpio\_path=/sys/class/gpio

local pin\_path=$gpio\_path/gpio$pin

case $verb in

read)

cat $pin\_path/value

;;

write)

echo $value > $pin\_path/value

;;

mode)

if [ ! -e $pin\_path ]; then

echo $pin > $gpio\_path/export

fi

echo $value > $pin\_path/direction

;;

state)

if [ -e $pin\_path ]; then

local dir=$(cat $pin\_path/direction)

local val=$(cat $pin\_path/value)

echo "$dir $val"

fi

;;

\*)

echo "Control the GPIO pins of the Raspberry Pi"

echo "Usage: $0 mode [pin] [in|out]"

echo " $0 read [pin]"

echo " $0 write [pin] [0|1]"

echo " $0 state [pin]"

echo "If GPIO\_PINS is an environment variable containing"

echo "a space-delimited list of integers, then up to 17"

echo "logical pins (0-16) will map to the physical pins"

echo "specified in the list."

;;

esac

}

# Just invoke our function if the script is called directly

if [ "$BASH\_SOURCE" == "$0" ]; then

gpio $@

fi

#Caller

#!/bin/bash

if [ “${user}” = “root”]; then

COUNTER=0

./gpio write $1 0

./gpio write $2 0

while [ $COUNTER -lt 10];do

#Execute GPIO code.

./gpio mode $1 out

./gpio write $1 1

sleep 5

./gpio write $1 0

./gpio mode $2 out

./gpio write $2 1

sleep 5

./gpio write $2 0

let COUNTER=COUNTER+1

done

fi

exit 0