Device Drivers Development Technical Report

Controlling Projector using Raspberry Pi

Sourabh Patel (MT2016159)

November 22 2017

Contents

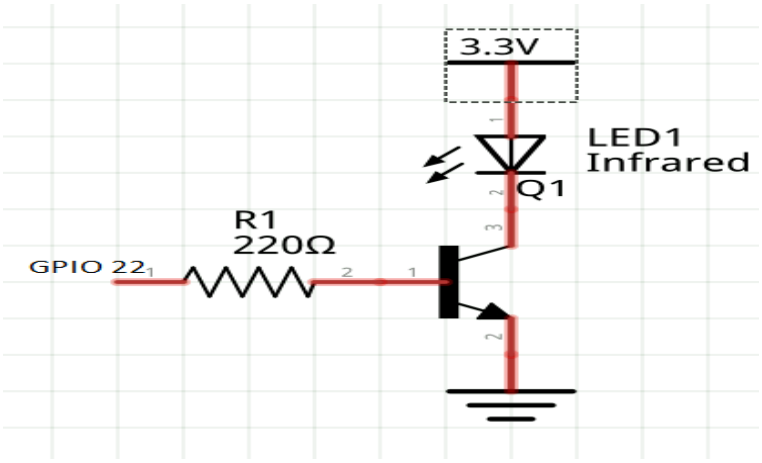
1 Abstract

In this project we have built a setup that demonstrates how a projector can be switched on or off from a Raspberry pi. Also the setup can turn off the projector automatically when HDMI cable is unplugged from the system. Also the setup decodes signal coming from projector remote by connecting IR receiver.

2 Introduction

Generally, everything that can send or receive infrared signals can be supported by LIRC. LIRC stands for Linux Infrared Remote Control. LIRC is a package that allows you to decode and send infra-red signals of many commonly used remote controls[3].The most important part of LIRC is the lircd daemon which decodes IR signals received by the device drivers and provides the information on a socket. It also accepts commands for IR signals.In our system we will be using lirc for receiving and sending ir signals to the projector.Also we will be using UDEV event mechanism. Using UDEV we can emit signals to switch off the projector receiver on detecting that HDMI is disconnected.We have used mailbox for communication between gpu and cpu as HDMI connector is controlled by GPU and our code will run at CPU side.There is some mechanism needed to find out whether HDMI is connected or not. For hardware configuration, we have used universal GPIO (general purpose input/output) board (for receiving the IR signals) connected through raspberry pi's GPIO pins, and we made a circuit for emitter using a 1k ohm resister, a PNP transister and an emitter (IR LED).

3 Circuit Diagram



4 Setting up LIRC in Raspberry Pi

We followed all the steps as given in [1].

First, we'll need to install and configure LIRC to run on the RaspberryPi using command

- sudo apt-get install lirc
- add
  lirc_dev, lirc_rpi
  gpio_in_pin=23 gpio_out_pin=22
  in /etc/modules file so that the kernel modules can be loaded automatically at boot time.
- change file /etc/lirc/hardware.conf to
  # /etc/lirc/hardware.conf
  #
  # Arguments which will be used when launching lircd
  LIRCD_ARGS=""

  #Don't start lircmd even if there seems to be a good config file
  #START_LIRCMD=false

  #Try to load appropriate kernel modules
  LOAD_MODULES=true

  # Run "lircd –driver=help" for a list of supported drivers.
  DRIVER=""
  # If DEVICE is set to /dev/lirc and devfs is in use /dev/lirc/0 will be
  # automatically used instead
  DEVICE=""
  MODULES="lirc_mceusb"

  # Default configuration files for your hardware if any
  LIRCD_CONF=""
  LIRCMD_CONF=""
  then reboot the raspberry pi so that all the changes in config.txt gets reflected.
- Now restart lircd so it picks up these changes:
  sudo /etc/init.d/lircd stop
  sudo /etc/init.d/lircd start
- update the configuration file of pi present in /boot/config.txt. This file contains all the parameters which are to be initialized at boot time .
  dtoverlay=lirc-rpi,gpio_in_pin=23,gpio_out_pin=22

- Testing the IR receiver
  sudo /etc/init.d/lircd stop
  mode2 -d /dev/lirc0
  You need to run mode2 only after stopping your lircd service

```
pi@raspberrypi:~ $ mode2 -d /dev/lirc0
Using driver default on device /dev/lirc0
Trying device: /dev/lirc0
Using device: /dev/lirc0
space 16777215
pulse 8909
space 4477
pulse 551
space 585
pulse 562
space 584
pulse 582
space 1668
pulse 551
```

4.1 Problems faced in configuring lirc

- The very first problem faced of getting internet connection in raspberry pi. We figured out a way to share internet connection from our computer with pi.
- When outputting raw data from IR receiver using mode2, I was getting continuous output even without pressing the buttons of projector remote. Figured out that it happened because we did not made the ground common for two circuits, the ir receiver and raspberry pi.
- When I ran mode2, I got an error that device is busy.This was because I ran mode2 on another terminal so mode2 was not able to open /dev/lirc0 file as it was already in use.

5 Testing the IR LED

In order to test ir led, we need a LIRC config file. Create a new remotecontrol configuration file using

irrecord -d /dev/lirc0 /lircd.conf

This program will record the signals from our remote control and create a config

```
# Type of device controlled
#      (TV, VCR, Audio, DVD, Satellite, Cable, HTPC, ...) :
# Device(s) controlled by this remote:

begin remote

  name  please
  bits           32
  flags SPACE_ENC|CONST_LENGTH
  eps            30
  aeps          100

  header       9058  4440
  one           586  1660
  zero          586   547
  ptrail        587
  repeat       9061  2220
  gap          107912
  toggle_bit_mask 0x0
  frequency    38000

      begin codes
          KEY_POWER               0x212F20DF 0x7E837DFC
          KEY_UP                  0x212FC23D 0x7E837DFC
      end codes

end remote
```

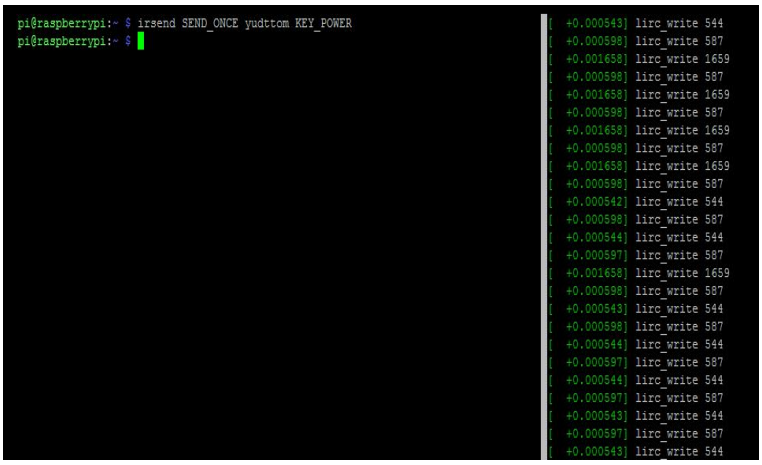
Remote configuration file

file for lircd. If file is not specified it defaults to "irrecord.lircd.conf". If file already exists and contains a valid config irrecord will use the protocol description found there and will only try to record the buttons.

Irrecord can be used with -force option to get raw code then use -a on the output to get hexadecimal code for that.

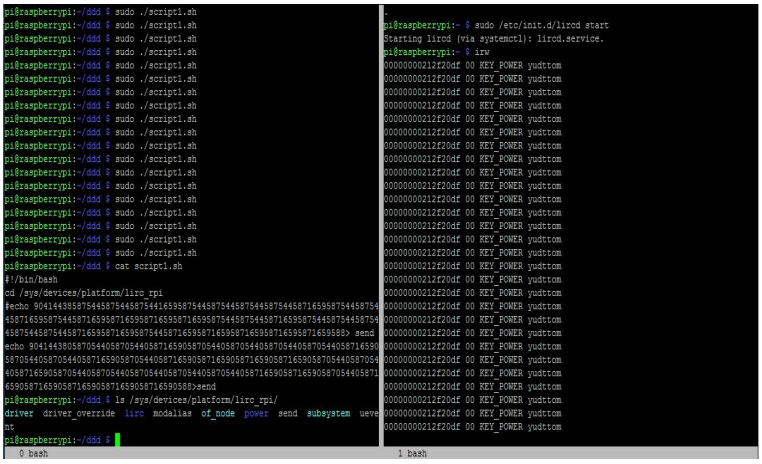
6 Updating lirc_rpi driver to use new gpio subsystem

Firstly we added some print statements in the lirc_rpi.c to see what was actually being sent from user space utility irsend. irsend SEND_ONCE yudttom KEY_POWER
We found out that irsend was writing raw code to the lirc device.



Output of dmesg for lirc_rpi module

Seeing from the above observations , we tried to make a sysfs attribute send where the input can be echoed and it would send it using the transmitter.The send file created under /sys/devices/platform/lirc_rpi/. The script1.sh file has a command to echo the raw code for power key in the send attribute file. Whenever the send attribute file is modified then send_store function of lirc_rpi gets called. In this function the input send which is of the form hexadecimal is converted to binary form. After that the signals pulse and space using irsend gets transmitted depending on the bit of the binary form whether it is 1 or 0. Whenever the scripts1.sh is executed , we are able to receive using irw.



Output of irw by runninf script1.sh

Earlier we were sending raw data in send attribute file.But after that we realized a pattern in the raw data which was being sent.We analyzed that all code

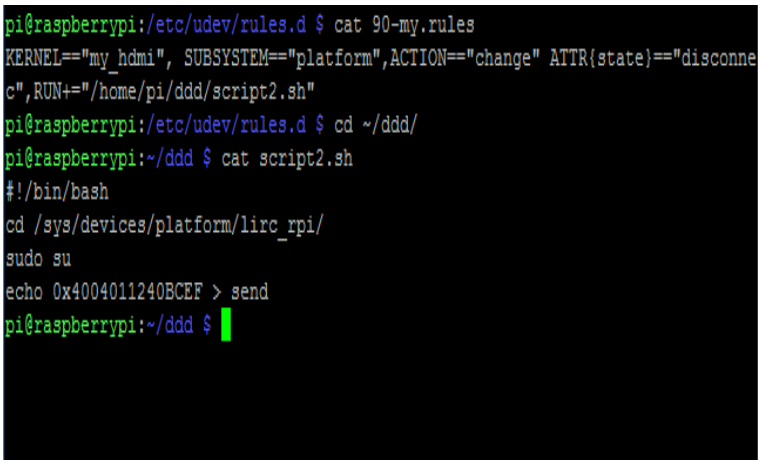
started with header followed by data and a trailing bit.As for now we have hard coded the values for header, pulse and space values for 0 and 1 and trailing bits.This part was working well when we tested it in the lab but it didn't work with classroom projector. It was because the code for the projector in classroom was very large which was outside the range of int. So instead of using int we converted the hex code into array of character.

7 Detecting HDMI

In order to detect whether hdmi is connnected or not, we looked into the device tree source of raspberry pi in /arch/arm/boot/dts/bcm2835-rpi-b-plus.dts[2]

```
    &hdmi {
hpd-gpios = <&gpio 46 GPIO_ACTIVE_LOW>;
};
```

From here we found out that hdmi port can be monitored using gpio pin 46. We exported the gpio46 using the file /sys/class/gpio/export and looked for any change. We observed that whenever hdmi cable is connected to port, it's value changed from 1 to 0. Initially we tried to monitor the gpio46 using interrupt handlers but that didn't work out well so we switched to polling. We have set a timer of 1 second after which the gpio46 is polled and monitored whether the state is changed or not. Also we were keeping track of the previous state. Whenever the state changes from 0 to 1, it states that the hdmi is disconnected[4]. It will generate the udev event. Then the kernel will read it and that's where udev rules will come into picture. The rules will call the script2.sh. Script2 contains the code for signalling the projector to power off by echoing the code of POWER_OFF in the send attribute.



Rules and script2.sh

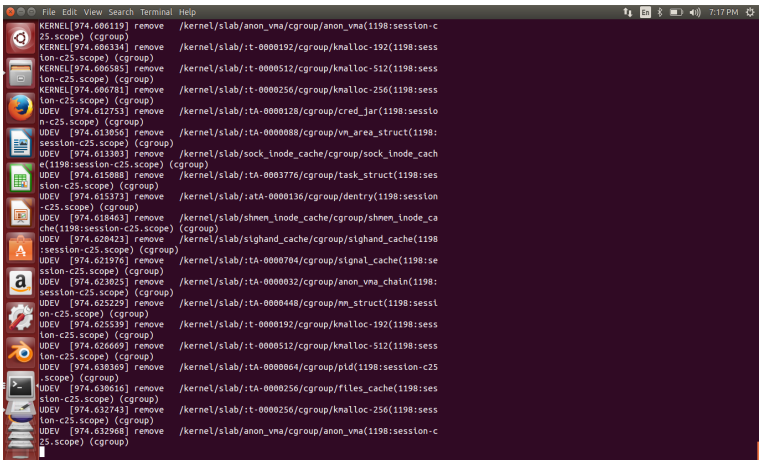


Figure shows the output of udevadm monitor

8 Future Scope

As for now we have hardcoded the values of header pulse, space, pulses and spaces for 0's and 1's in our lirc module but later we can add attributes to show and store the values from sysfs so that our module will much more flexible. Moreover for the second module we are reading the name of the device connected and comparing it with the name of the class projector, we can add an attribute to take the name of the projector from sysfs itself.

References

[1] Setting up lirc on raspberry pi: `http://www-cs-faculty.stanford.edu/~uno/abcde.html`

[2] device tree of pi: ,`https://elixir.free-electrons.com/linux/v4.9.44/source/arch/arm/boot/dts/bcm2835-rpi-b-plus.dts`

[3] lirc documentation : ,`http://www.lirc.org/`

[4] Mailbox: ,`https://github.com/raspberrypi/firmware/wiki/Mailboxes`