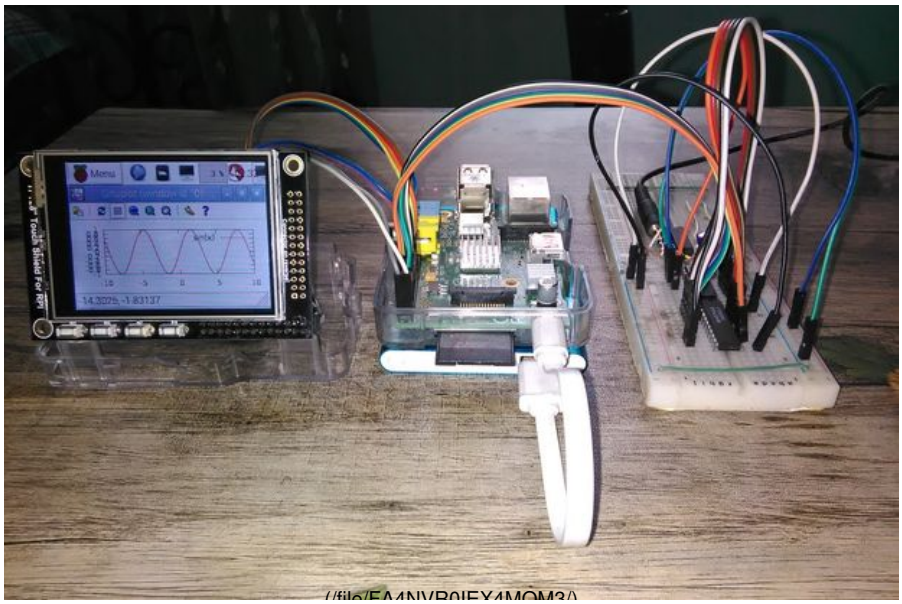
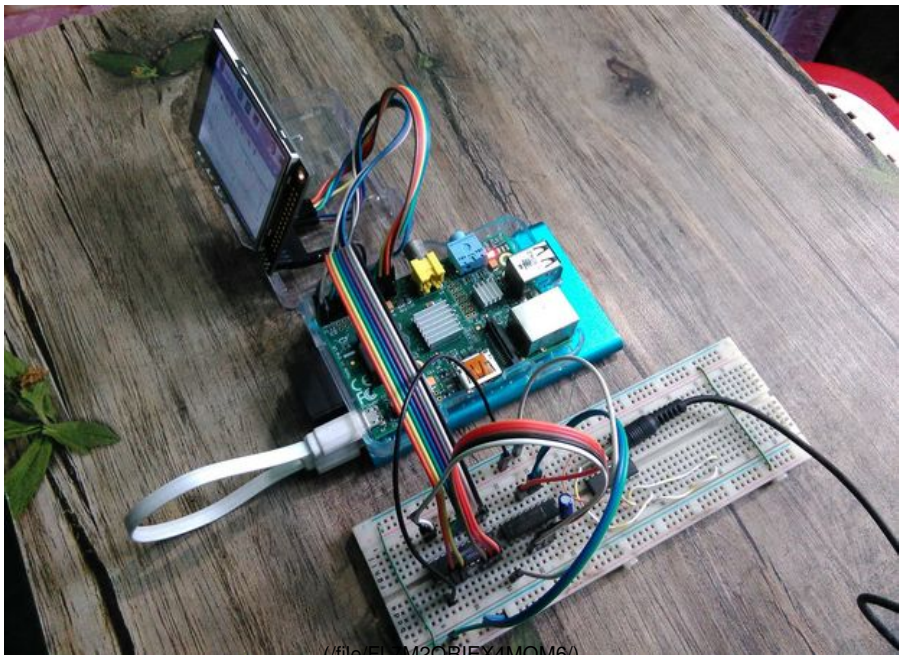




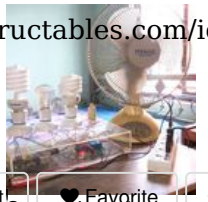
An oscilloscope is a laboratory instrument commonly used to display and analyze the waveform of electronic signals. In effect, the device draws a graph of the instantaneous signal voltage as a function of time.

Oscilloscopes are used in the sciences, medicine, engineering, and telecommunications industry. Oscilloscopes are very essential and best friend for students, maker, hobbyist and electronics enthusiast. While a digital

PiScope (Raspberry Pi based Oscilloscope)

taifur (/member/taifur/) in raspberry pi (/tag/type-id/category-technology/channel-raspberry-pi/)

multimeter can help you measure steady state and RMS (Root-Mean-Square) voltages, the oscilloscope can not only measure peak-to-peak voltages, but more importantly provide timing information on your signal. For instance, have you ever been working with an Arduino controlling a servo motor that has to have just the right pulse width modulation in order to spin clockwise instead of counter-clockwise? During your programming, you may have wondered just how close the pulse width was to what was needed. With an oscilloscope you can measure these pulses. When dealing with analog signals, you can use an oscilloscope to see how close you are to the frequency you need or measure what frequency you need to filter. With so many digital electronic projects, timing between signals is extremely important. Therefore, having an oscilloscope is essential but unfortunately they are very expensive.



(/id/Voice-Controlled-Smart-Home/)

You can find several DIY oscilloscope in Internet and some links are provided below:

1. PC sound card based oscilloscope

http://homediyelectronics.com/projects/howtomakeafreesoundcardpcoscilloscope/ (http://homediyelectronics.com/projects/howtomakeafreesoundcardpcoscilloscope/)

https://www.instructables.com/id/Use-Your-Laptop-as-Oscilloscope/ (https://www.instructables.com/id/Use-Your-Laptop-as-Oscilloscope/)

http://makezine.com/projects/sound-card-oscilloscope/ (http://makezine.com/projects/sound-card-oscilloscope/)

2. Arduino based oscilloscope

https://www.instructables.com/id/Arduino-Oscilloscope-poor-mans-Oscilloscope/ (https://www.instructables.com/id/Arduino-Oscilloscope-poor-mans-Oscilloscope/)

http://www.miupanel.com/Projects/Arduino-Advanced-Oscilloscope (http://www.miupanel.com/Projects/Arduino-Advanced-Oscilloscope)

https://www.instructables.com/id/Girino-Fast-Arduino-Oscilloscope/ (https://www.instructables.com/id/Girino-Fast-Arduino-Oscilloscope/)

https://www.instructables.com/id/DIY-USB-OSCILLOSCOPE-IN-A-MATCHBOX/ (https://www.instructables.com/id/DIY-USB-OSCILLOSCOPE-IN-A-MATCHBOX/)

3. CRT TV based oscilloscope

https://www.instructables.com/id/Fully-Functional-Television-Oscilloscope/ (https://www.instructables.com/id/Fully-Functional-Television-Oscilloscope/)

https://www.instructables.com/id/Mini-TV-Oscilloscope/ (https://www.instructables.com/id/Mini-TV-Oscilloscope/)

https://www.instructables.com/id/How-To-Make-A-CRT-TV-Into-an-Oscilloscope/ (https://www.instructables.com/id/How-To-Make-A-CRT-TV-Into-an-Oscilloscope/)

4. Smart phone based oscilloscope

https://www.instructables.com/id/OscilloPhone-Use-your-Smartphone-as-an-Oscilloscop/ (https://www.instructables.com/id/OscilloPhone-Use-your-Smartphone-as-an-Oscilloscop/)

https://www.instructables.com/id/A-Preamplifier-for-Smartphone-Oscilloscopes/ (https://www.instructables.com/id/A-Preamplifier-for-Smartphone-Oscilloscopes/)

http://projectproto.blogspot.com/2010/09/android-bluetooth-oscilloscope.html (http://projectproto.blogspot.com/2010/09/android-bluetooth-oscilloscope.html)

5. Raspberry Pi based oscilloscope

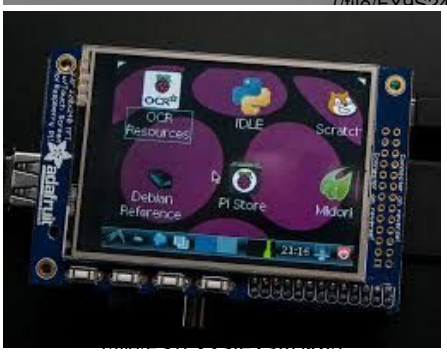
https://www.instructables.com/id/PiMSO-A-Raspberry-Pi-based-Wi-Fi-Oscilloscope/ (https://www.instructables.com/id/PiMSO-A-Raspberry-Pi-based-

PiScope (Raspberry Pi based Oscilloscope)

by taifur (/member/taifur/) in raspberry-pi (/tag/type-id/category-technology/channel-raspberry-pi/)

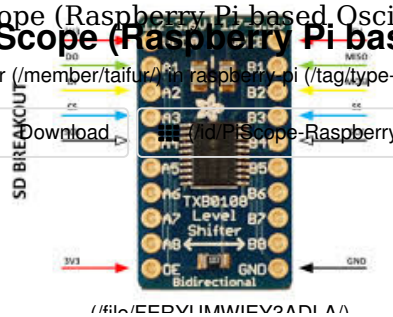
All of those oscilloscopes have their own pros and cons. Most PC and Arduino based oscilloscope can not sample more than several kilohertz. Sometimes PC based oscilloscope can burn your computer motherboard. Previous Raspberry Pi based oscilloscope required special hardware. I will show you step by step guide how to build a Raspberry Pi based oscilloscope without special hardware. I would like to thank Mr. Daniel Pelikan who first published the idea in the MagPi magazine, Issue 24 (https://www.raspberrypi.org/magpi/issues/24).

Step 1: Required Component for PiScope



PiScope (Raspberry Pi based Oscilloscope)

by taifur (/member/taifur/) in raspberry-pi (/tag/type-id/category-technology/channel-raspberry-pi/)



Hardware

- Raspberry Pi
- SD Card with Raspbian image
- Adafruit PiTFT (Touchscreen display)
- Breadboard
- CA3306 (6 bit parallel A/D converter IC)
- TXB0108 (logic level converter IC)
- Jumper Wire
- Access to a computer

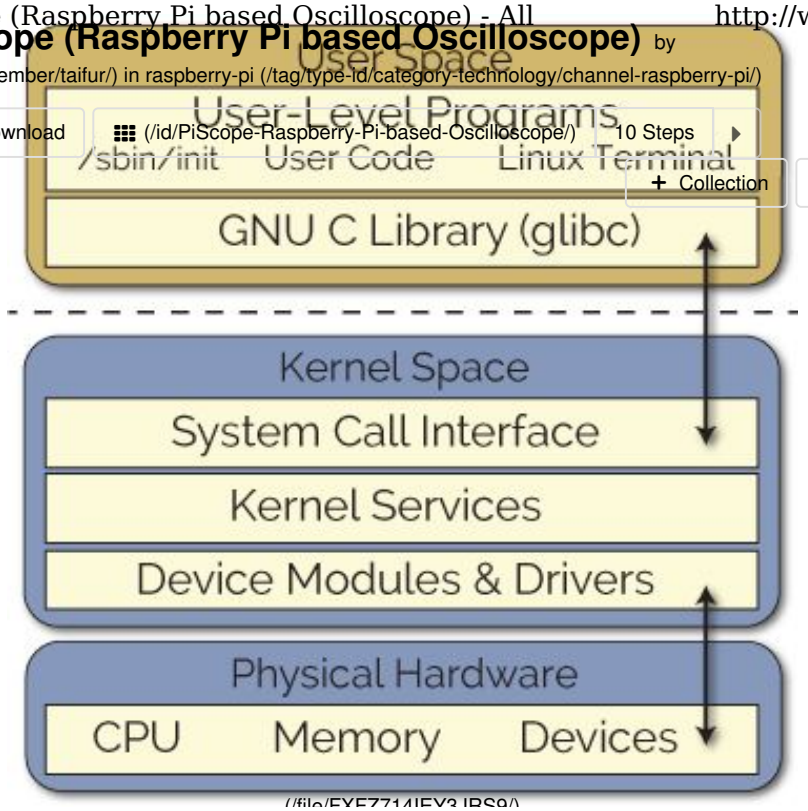
Software

- SSH Client (I used PuTTY (http://www.putty.org/))
- FTP Client (I used FileZilla (https://filezilla-project.org/))

Step 2: Setup your Raspberry Pi

PiScope (Raspberry Pi based Oscilloscope) by

taifur (/member/taifur/) in raspberry-pi (/tag/type-id/category-technology/channel-raspberry-pi/)



The Raspberry Pi is a general purpose single board computer that can run a Linux operation system. However, Linux operating systems do not normally run processes in realtime. This is because the operating system listens for inputs from other devices, rather than just processing one command at a time. A parallel ADC is used to take input and when reading an external ADC, one needs to make sure that the time between each sample point is the same. Without a realtime operating system, this is not guaranteed. Special thanks to Mr. Daniel Pelikan again for writing a Linux kernel module to solve the problem.

What is a Linux kernel module?

A loadable kernel module (LKM) is a mechanism for adding code to, or removing code from, the Linux kernel at run time. Modules are pieces of code that can be loaded and unloaded into the kernel upon demand. They extend the functionality of the kernel without the need to reboot the system. For example, one type of module is the device driver, which allows the kernel to access hardware connected to the system. Without this modular capability, the Linux kernel would be very large, as it would have to support every driver that would ever be needed on the BBB. You would also have to rebuild the kernel every time you wanted to add new hardware or update a device driver. The downside of LKMs is that driver files have to be maintained for each device. LKMs are loaded at run time, but they do not execute in user space — they are essentially part of the kernel. To learn more about Linux kernel module follow the links given below.

http://www.tldp.org/LDP/lkmpg/2.6/html/ (http://www.tldp.org/LDP/lkmpg/2.6/html/)

http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction/ (http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction/)

http://www.thegeekstuff.com/2013/07/write-linux-kernel-module/ (http://www.thegeekstuff.com/2013/07/write-linux-kernel-module/)

Writing a Linux kernel module provides the possibility to perform low level hardware operations. We need to run with the highest possible priority, reading the GPIO register with the system interrupts disabled for as short a time as possible.

PiScope (Raspberry Pi based Oscilloscope)

A parallel ADC can be used to take a sample on the rising edge of a clock signal and output the sample on the data pins on the falling edge. The aim is to clock the ADC at our required sample rate and read all of the data pins between each sample.

Step 3: Compiling a Kernel Module

After writing the kernel module you have to compile it. Kernel modules need to be compiled a bit differently from regular userspace apps. Former kernel versions required us to care much about these settings, which are usually stored in Makefiles. Although hierarchically organized, many redundant settings accumulated in sub level Makefiles and made them large and rather difficult to maintain. Fortunately, there is a new way of doing these things, called kbuild, and the build process for external loadable modules is now fully integrated into the standard kernel build mechanism. For that we need to prepare our developing environment. Kernel modules need to be compiled with certain gcc options to make them work. In addition, they also need to be compiled with certain symbols defined.

For compiling Linux kernel module two possible routes available:

1. Compile on the Raspberry Pi itself

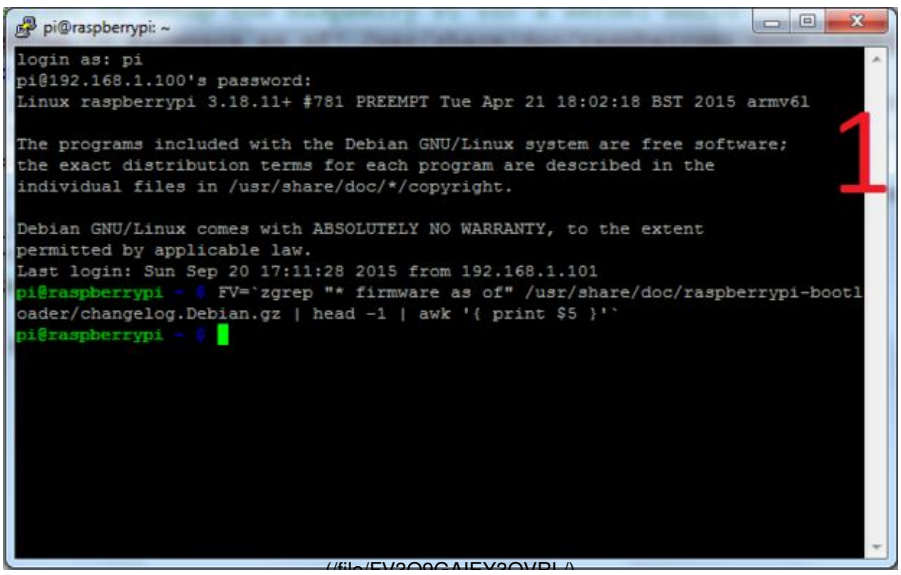
2. Cross compile on another Linux system

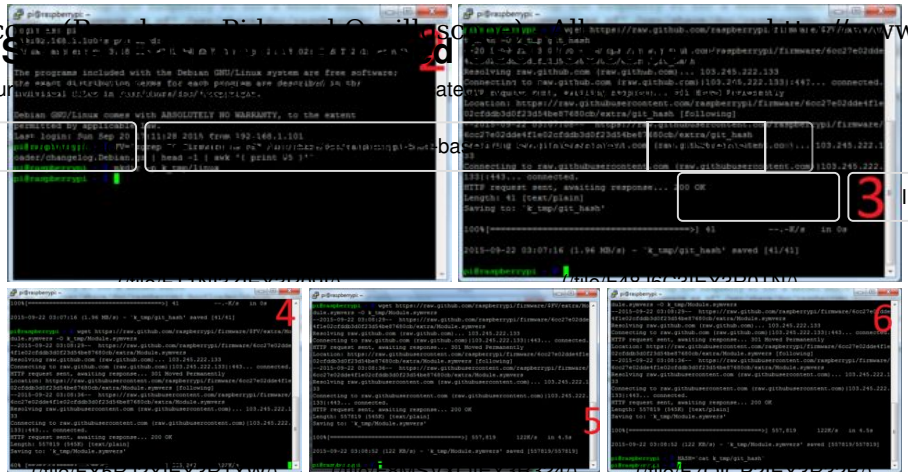
I will show here the first one, though takes some more time but requires less setup compare to cross compilation. For more about kernel compilation follow the link:

http://elinux.org/Raspberry_Pi_Kernel_Compilation (http://elinux.org/Raspberry_Pi_Kernel_Compilation)

Our process will take about 30 minutes to complete the process.

Step 4: Set up Raspberry Pi build environment





Show All Items

I Made it! Favorite Share

In order to build a kernel module, we need the kernel headers (or kernel source) that match the binary image. The headers provide vital definitions that are needed in order to compile the source code for the module. Also, Linux performs a safety check called “version matching” when it loads a finished kernel module. The kernel version and module version must match or, at best, Linux complains, or, at worst, Linux refuses to load the module.

To prepare your raspberry pi to compile kernel module correctly follow the steps. First connect your raspberry pi to your computer using SSH (details of connecting Raspberry pi using PuTTY (https://www.raspberrypi.org /documentation/remote-access/ssh/windows.md)). Then type the following

```
FV=`zgrep "* firmware as of" /usr/share/doc/raspberrypi-bootloader/change
log.Debian.gz | head -1 | awk '{ print $5 }'`
```

Make a directory k_tmp/linux by the command

```
mkdir -p k_tmp/linux
```

Download firmware from github.com

```
wget https://raw.github.com/raspberrypi/firmware/$FV/extra/git_hash -O k_
tmp/git_hash
```

The build directory must have a copy of the matching module version information before building. Module version information is stored in a file named /usr/src/linux/Module.symvers. This file is created during the kernel build — a process that would take 10+ hours on the Raspberry Pi. Fortunately, Module.symvers can also be downloaded from github.com. Download Module.symvers file

```
wget https://raw.github.com/raspberrypi/firmware/$FV/extra/Module.symvers
-O k_tmp/Module.symvers
```

Download linux source code. The compressed source code is about 110MBytes in size. This is the full source for the kernel including all of the headers.

```
HASH=`cat k_tmp/git_hash`
wget -c https://github.com/raspberrypi/linux/tarball/$HASH -O k_tmp/linux
.tar.gz
```

Goto the created directory k_tmp

```
cd k_tmp
```

Decompresses and unpacks the source code in the compressed TAR file

PiScope (Raspberry Pi based Oscilloscope) by taifur (/member/taifur/) in raspberry-pi (/tag/type-id/category-technology/channel-raspberry-pi/)

Move it to the src directory

```
KV=`uname -r`
sudo mv raspberrypi-linux* /usr/src/linux-source-$KV
```

Type and run the following commands

```
sudo ln -s /usr/src/linux-source-$KV /lib/modules/$KV/build
```

```
sudo cp Module.symvers /usr/src/linux-source-$KV/
```

```
sudo zcat /proc/config.gz > /usr/src/linux-source-$KV/.config
```

change the directory

```
cd /usr/src/linux-source-$KV/
```

Run oldconfig

```
sudo make oldconfig
```

Set up the source tree to build kernel modules

```
sudo make prepare
```

Run make script

```
sudo make scripts
```

If everything done correctly your Raspberry Pi is now ready to compile kernel module.

You can run all the command from a single bash script, for that make a script and open it using nano

sudo nano piscript.sh

and type the following command

```
#! /bin/bash
# A script to setup the Raspberry Pi for a kernel build
FV=`zgrep "* firmware as of" /usr/share/doc/raspberrypi-bootloader/change
log.Debian.gz | head -1 | awk '{ print $5 }'`
mkdir -p k_tmp/linux
wget https://raw.github.com/raspberrypi/firmware/$FV/extra/git_hash -O k_
tmp/git_hash
wget https://raw.github.com/raspberrypi/firmware/$FV/extra/Module.symvers
-O k_tmp/Module.symvers
HASH=`cat k_tmp/git_hash`
wget -c https://github.com/raspberrypi/linux/tarball/$HASH -O k_tmp/linux
.tar.gz
cd k_tmp
tar -xzf linux.tar.gz
KV=`uname -r`

sudo mv raspberrypi-linux* /usr/src/linux-source-$KV
sudo ln -s /usr/src/linux-source-$KV /lib/modules/$KV/build
sudo cp Module.symvers /usr/src/linux-source-$KV/
sudo zcat /proc/config.gz > /usr/src/linux-source-$KV/.config
cd /usr/src/linux-source-$KV/
sudo make oldconfig
sudo make prepare
sudo make scripts
```

Make the file executable using the command

```
sudo chmod +x piscope.sh
```

Run the file using

PiScope (Raspberry Pi based Oscilloscope) by

(/member/taifur/)

taifur (/member/taifur/) in raspberry-pi (/tag/type-id/category-technology/channel-raspberry-pi/)

Download (/id/PiScope-Raspberry-Pi-based-Oscilloscope/) 10 Steps

Done!

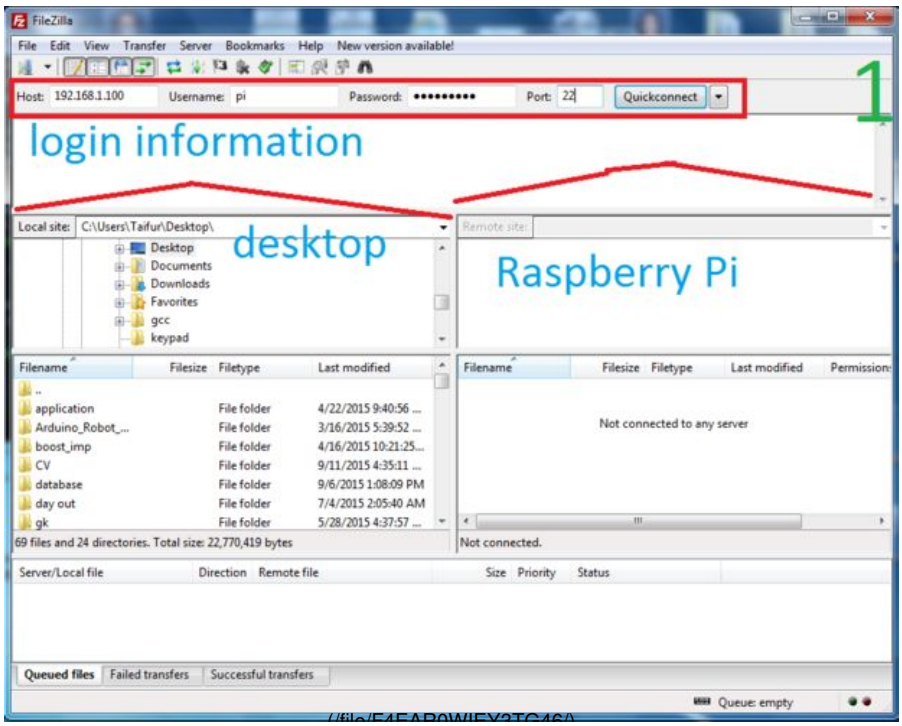
You can also upload the file piscope.sh to Raspberry Pi using FTP client.

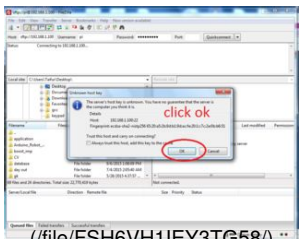
Collection I Made it! Favorite Share

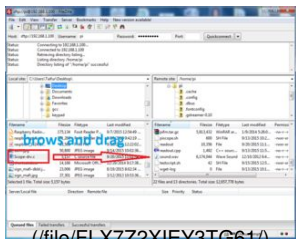
piscope.sh (https://cdn.instructables.com/ORIG/FZ7/Z9WD/IEY3SDZH/FZ7Z9WDIEY3SDZH.sh)

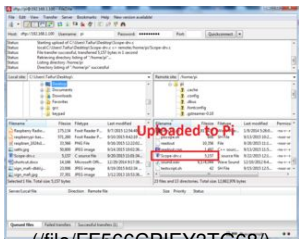
/FZ7Z9WDIEY3SDZH.sh)

Step 5: Transferring Files from your Computer to Raspberry Pi Using FTP Client









Transferring Files to and from the Raspberry Pi

PiScope (Raspberry Pi based Oscilloscope)

Typing program using nano is not a easy task. So you can make your file in your pc and upload it to your Raspberry Pi using FTP client. I will show you how to upload a file using a FileZilla FTP client. Download FileZilla (https://filezilla-project.org/) and install it to your pc. Run FileZilla and enter your Pi IP address to the Host box, enter pi as user name and raspberry as password and 22 as Port, then click to quickconnect. After successful connection you see your desktop files and directories to the left window and pi's files and directories to the right windows. Brows the file of your desktop from the left window and drag it to the right window. It will upload the selected file to raspberry pi home directory. All steps are shown in figures attached herewith.

Details: http://trevorappleton.blogspot.com/2014/03/remotely-copy-files-to-and-from-your.html (http://trevorappleton.blogspot.com/2014/03/remotely-copy-files-to-and-from-your.html)

Step 6: Writing the Kernel Module

Create a C file called Scope-drv.c that contains the following code

PiScope (Raspberry Pi based Oscilloscope) by

taifur (/member/taifur/) in raspberry-pi (/tag/type-id/category-technology/channel-raspberry-pi/)


```
#include<linux/kernel.h>
#include<linux/module.h>
#include<linux/init.h>
#include<asm/uaccess.h>
#include<linux/time.h>
#include<linux/io.h>
#include<linux/vmalloc.h>

int init_module(void);
void cleanup_module(void);
static int device_open(struct inode *, struct file *);
static int device_release(struct inode *, struct file *);
static ssize_t device_read(struct file *, char *, size_t, loff_t *);
static ssize_t device_write(struct file *, const char *, size_t, loff_t *
);

#define SUCCESS 0
#define DEVICE_NAME "chardev"
#define BUF_LEN 80

/* setting and macros for the GPIO connections */
#define BCM2708_PERI_BASE 0x20000000
#define GPIO_BASE (BCM2708_PERI_BASE + 0x20000000)

#define INP_GPIO(g) *(gpio.addr + ((g)/10)) &= ~(7<<(((g)%10)*3))
#define SET_GPIO_ALT(g,a) *(gpio.addr + (((g)/10))) |= (((a)<=3?(a) + 4:(
a)==4?3:2)<<(((g)%10)*3))

/* GPIO clock */
#define CLOCK_BASE  (BCM2708_PERI_BASE + 0x00101000)
#define GZ_CLK_BUSY  (1 << 7)

/* Number of samples to capture */
#define SAMPLE_SIZE 10000

/* Define GPIO pins */
/* ADC 1 */
#define BIT0_PIN 7
#define BIT1_PIN 8
#define BIT2_PIN 9
#define BIT3_PIN 10
#define BIT4_PIN 11
#define BIT5_PIN 25

/* ADC 2 */
#define BIT0_PIN2 17
#define BIT1_PIN2 18
#define BIT2_PIN2 22
#define BIT3_PIN2 23
#define BIT4_PIN2 24
#define BIT5_PIN2 27

struct bcm2835_peripheral {
        unsigned long addr_p;
        int mem_fd;
        void *map;
        volatile unsigned int *addr;

};

static int map_peripheral(struct bcm2835_peripheral *p);
static void unmap_peripheral(struct bcm2835_peripheral *p);
static void readScope(void);

static int Major;
static int Device_Open = 0;
static char msg[BUF_LEN];
static char *msg_Ptr;

static unsigned char *buf_p;

static struct file_operations fops = {
```

PiScope (Raspberry Pi based Oscilloscope) by

taifur (/member/taifur/) in raspberry-pi (/tag/type-id/category-technology/channel-raspberry-pi/)


```
        read = device_read
        write = device_write
        open = device_open
        release = device_release
};

static struct bcm2835_peripheral myclock = {CLOCK_BASE};
static struct bcm2835_peripheral gpio = {GPIO_BASE};
struct DataStruct{
        uint32_t Buffer[SAMPLE_SIZE];
        uint32_t time;
};

struct DataStruct dataStruct;

static unsigned char *ScopeBufferStart;
static unsigned char *ScopeBufferStop;

static int map_peripheral(struct bcm2835_peripheral *p){
        p->addr=(uint32_t *)ioremap(GPIO_BASE, 41*4);
        return 0;
}

static void unmap_peripheral(struct bcm2835_peripheral *p){
        iounmap(p->addr);
}

static void readScope(){
        int counter=0;
        struct timespec ts_start, ts_stop;

        local_irq_disable();
        local_fiq_disable();

        getnstimeofday(&ts_start);

        while(counter<SAMPLE_SIZE){
                dataStruct.Buffer[counter++]= *(gpio.addr + 13);
        }
    getnstimeofday(&ts_stop);

        local_fiq_enable();
        local_irq_enable();

        dataStruct.time = timespec_to_ns(&ts_stop) - timespec_to_ns(&ts_s
tart);

        buf_p = (unsigned char*)&dataStruct;
        ScopeBufferStart = (unsigned char*)&dataStruct;
        ScopeBufferStop = ScopeBufferStart + sizeof(struct DataStruct);
}

int init_module(void){
        struct bcm2835_peripheral *p=&myclock;
        int speed_id = 6;

        Major = register_chrdev(0, DEVICE_NAME, &fops);
        if(Major < 0){
                printk(KERN_ALERT "Reg. char dev fail %d\n",Major);
                return Major;
        }
        printk(KERN_INFO "Major number %d.\n", Major);
        printk(KERN_INFO "created a dev file with\n");
        printk(KERN_INFO "'mknod /dev/%s c %d 0'.\n", DEVICE_NAME, Major)
;

        if(map_peripheral(&gpio) == -1){
                printk(KERN_ALERT "Failed to map GPIO\n");
                return -1;
        }

        INP_GPIO(BIT0_PIN);
```

PiScope (Raspberry Pi based Oscilloscope) by

taifur (/member/taifur/) in raspberry-pi (/tag/type-id/category-technology/channel-raspberry-pi/)


```
        INP_GPIO(BIT1_PIN);
        INP_GPIO(BIT2_PIN);
        INP_GPIO(BIT3_PIN);
        INP_GPIO(BIT4_PIN);
        INP_GPIO(BIT5_PIN);

        INP_GPIO(BIT0_PIN2);
        INP_GPIO(BIT1_PIN2);
        INP_GPIO(BIT2_PIN2);
        INP_GPIO(BIT3_PIN2);
        INP_GPIO(BIT4_PIN2);
        INP_GPIO(BIT5_PIN2);

        /* set clock signal to pin 4 */
        p->addr=(uint32_t *)ioremap(CLOCK_BASE, 41*4);

        INP_GPIO(4);
        SET_GPIO_ALT(4,0);
        *(myclock.addr+28)=0x5A000000 | speed_id;

        while(*(myclock.addr+28) & GZ_CLK_BUSY){};

        *(myclock.addr+29)= 0x5A000000 | (0x32 << 12) | 0;

        *(myclock.addr+28)=0x5A000000 | speed_id;
```

The program contains some important functions. In order to make a kernel module work. the module needs some special entry functions. One of these function is the init_module(), which is called when the kernel module is loaded. The device_open() function is called when the device file associated with the kernel module is opened. Opening the device file causes the ADC to be read out 10,000 times, where the results are saved in memory. The device_release() function is called when the device is closed. The device_read() function is called when a process reads from the device file. This function returns the measurements that were made when the device file was opened. The last function device_write() is needed to handle the case when a process tries to write to the device file.

More about kernel module: http://www.tldp.org/LDP/lkmpg/2.6/html/lkmpg.html (http://www.tldp.org/LDP/lkmpg/2.6/html/lkmpg.html)

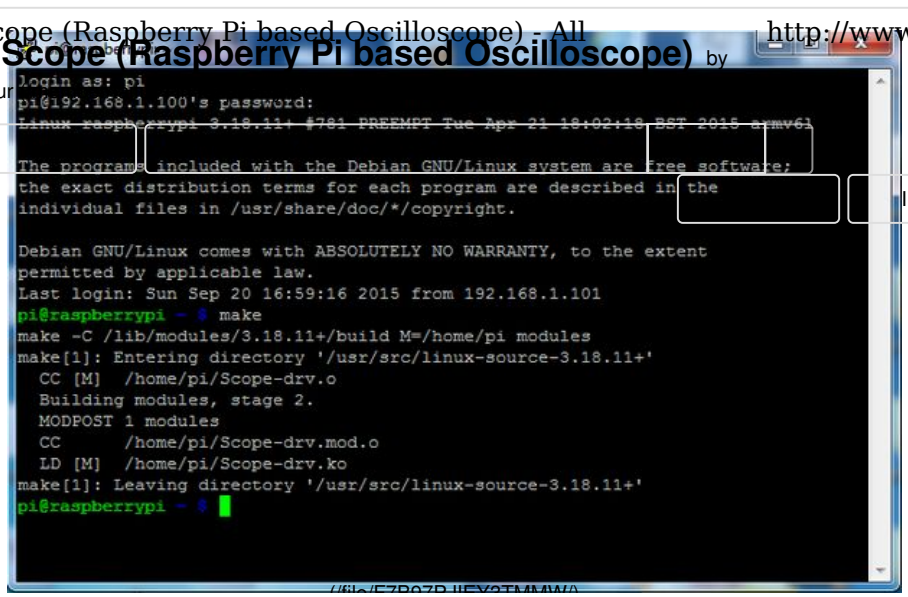
Full program is attached, you can upload it using FileZilla.

Scope-dry.c (https://cdn.instructables.com/ORIG/F70/4WVS/IEY3THMA/F704WVSIEY3THMA.c) wnload (https://cdn.instructables.com/ORIG/F70/4WVS/IEY3THMA/F704WVSIEY3THMA.c)

Makefile (https://www.instructables.com/files/orig/FJ8/9T3X/IEY3THNP/FJ89T3XIEY3THNP.null) oad (https://www.instructables.com/files/orig/FJ8/9T3X/IEY3THNP/FJ89T3XIEY3THNP.null)

Step 7: Building and loading the module

PiScope (Raspberry Pi based Oscilloscope) by taifur



Create a Makefile in the same directory as the Scope-drv. c file using command

```
sudo nano Makefile.sh
```

Your Makefile contains the following (indents should be a single tab)

```
<code>
obj-m += Scope-drv.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Now you can compile your module typing

```
make
```

Once the module has been successfully compiled, load the module by typing:

```
sudo insmod . /Scope-drv.ko
```

Then assign the device file, by typing:

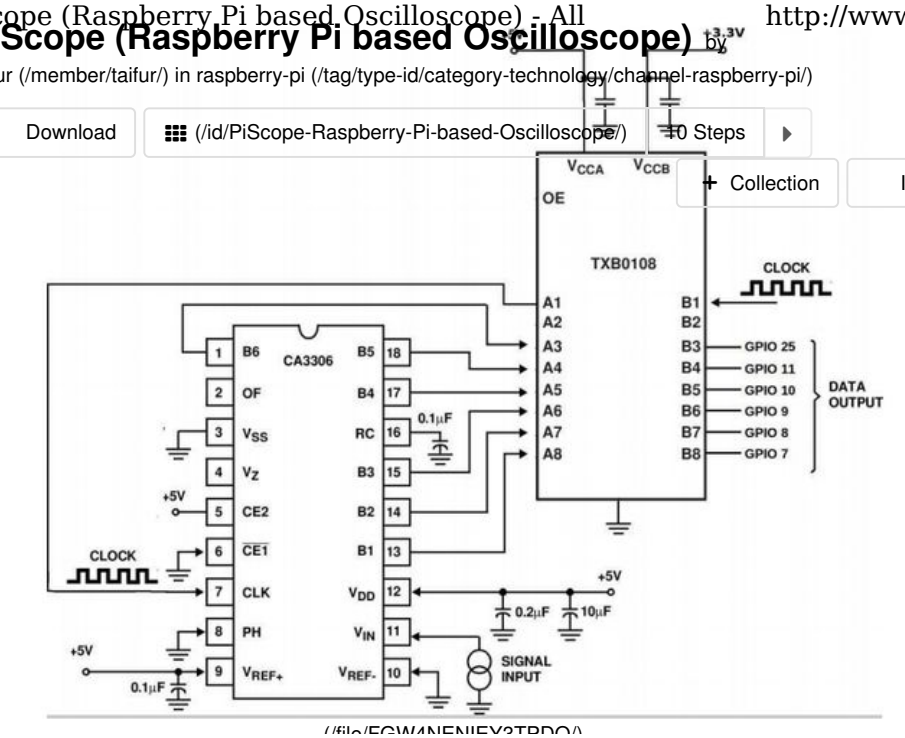
```
sudo mknod /dev/chardev c 248 0
```

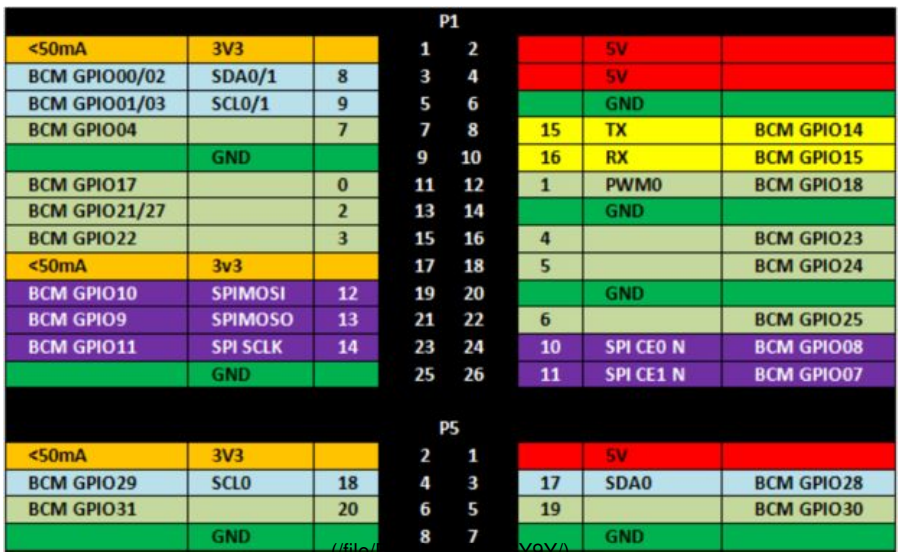
Makefile (https://www.instructables.com/files/orig/FDO/DT5Y/IEY3TL49/FDODT5YIEY3TL49.null)
ad (https://www.instructables.com/files/orig/FDO/DT5Y/IEY3TL49/FDODT5YIEY3TL49.null)
/FDODT5YIEY3TL49.null)

Step 8: Connecting to ADC

PiScope (Raspberry Pi based Oscilloscope) by

taifur (/member/taifur/) in raspberry-pi (/tag/type-id/category-technology/channel-raspberry-pi/)





P1							
<50mA	3V3		1	2		5V	
BCM GPIO00/02	SDA0/1	8	3	4		5V	
BCM GPIO01/03	SCL0/1	9	5	6		GND	
BCM GPIO04		7	7	8	15	TX	BCM GPIO14
	GND		9	10	16	RX	BCM GPIO15
BCM GPIO17		0	11	12	1	PWM0	BCM GPIO18
BCM GPIO21/27		2	13	14		GND	
BCM GPIO22		3	15	16	4		BCM GPIO23
<50mA	3v3		17	18	5		BCM GPIO24
BCM GPIO10	SPIMOSI	12	19	20		GND	
BCM GPIO9	SPIMOSO	13	21	22	6		BCM GPIO25
BCM GPIO11	SPI SCLK	14	23	24	10	SPI CE0 N	BCM GPIO08
	GND		25	26	11	SPI CE1 N	BCM GPIO07

P5							
<50mA	3V3		2	1		5V	
BCM GPIO29	SCL0	18	4	3	17	SDA0	BCM GPIO28
BCM GPIO31		20	6	5	19		BCM GPIO30
	GND		8	7		GND	

A parallel ADC can be used to take a sample on the rising edge of a clock signal and output the sample on the data pins on the falling edge. The aim is to clock the ADC at our required sample rate and read all of the data pins between each sample.

Our kernel module is ready now an ADC is needed to provide the input data. For this article, a CA3306 ADC from Intersil was used. This is a 6-bit 15 MSPS ADC with a parallel read out. This ADC is very cheap and fast. Many other ADC chips with parallel readout could be used, although it is necessary to check the datasheet for connection details and clock speed settings,etc..

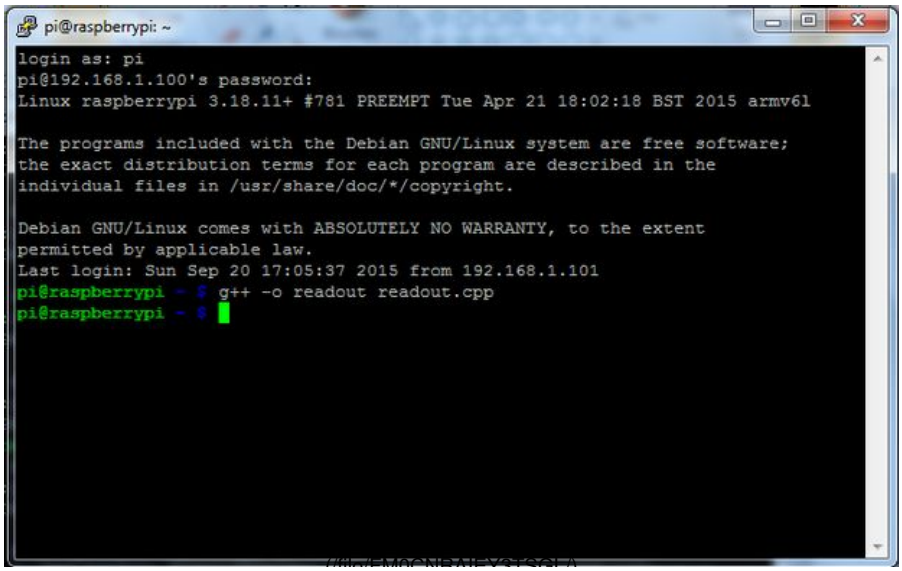
For the selected ADC, 6-bit implies that between the ground level (0V) and the reference voltage (5V) there are 64 divisions to represent the signal. This is quite course, but is enough for simple applications. The selected ADC operates with 5V logic, but the Raspberry Pi uses 3V3 logic. Therefore, a level converter is needed to protect the Raspberry Pi from being damaged. The simplest way to achieve this is to use a dedicated level converter, such as the TXB0108 from Texas Instruments. To ensure that stable readings are obtained from the ADC, it is recommended that a separate 5V supply is used as your VREF+ and VDD supply. This prevents voltage drops that can occur if the power supply is shared with the Raspberry Pi. However, a common ground (GND) connection should be used for the external supply,ADC and Raspberry Pi.

Circuit diagram is attached, connect accordingly to raspberry pi.

PiScope (Raspberry Pi based Oscilloscope) by

taifur (/member/taifur/) in raspberry-pi (/tag/type-id/category-technology/channel-raspberry-pi/)

Step 9: Data acquisition for PiScope



Once the ADC has been connected and the kernel module has been loaded, data can be read from the ADC by connecting to the device file associated with the kernel module. To connect to the kernel module, another program is needed. This program could be written in several different programming languages. For this article, C++ was chosen. Create a new file called readout.cpp and add following code or upload readout.cpp attached below.

PiScope (Raspberry Pi based Oscilloscope) by

taifur (/member/taifur/) in raspberry-pi (/tag/type-id/category-technology/channel-raspberry-pi/)


```
#include<iostream>
#include<cstdlib>
#include<fstream>
#include<bitset>

typedef unsigned int uint32_t;

const int DataPointsRPi=10000;
struct DataStructRPi{
        uint32_t Buffer[DataPointsRPi];
        uint32_t time;
};

int main(){
        struct DataStructRPi dataStruct;
        unsigned char *ScopeBufferStart;
        unsigned char *ScopeBufferStop;
        unsigned char *buf_p;

        buf_p=(unsigned char*)&dataStruct;
        ScopeBufferStart=(unsigned char*)&dataStruct;
        ScopeBufferStop=ScopeBufferStart+sizeof(struct DataStructRPi);

        std::string line;
        std::ifstream myfile ("/dev/chardev");
```

This program includes the definition of the data struct that matches the version in the kernel module. The main() function connects to the /dev/chardev device, which causes the kernel module to readout the ADC and store the values. Then the data are read from the memory buffer and copied into the local buffer within the main() function. Finally, the data are converted into a time in nano seconds and voltage values. The time and two voltage values are then printed in columns. The voltage values read by the ADCs are encoded as six bits. The bits are decoded using bit shift operations and bit wise and operations.

To compile the data acquisition program, type:

```
g++ -o readout readout.cpp
```

Then run the program by typing:

```
./readout > data.txt
```

The data file can be displayed using gnuplot. Install gnuplot by typing:

```
sudo apt-get install -y gnuplot-x11<br>
```

Then type gnuplot and enter the macro given below:

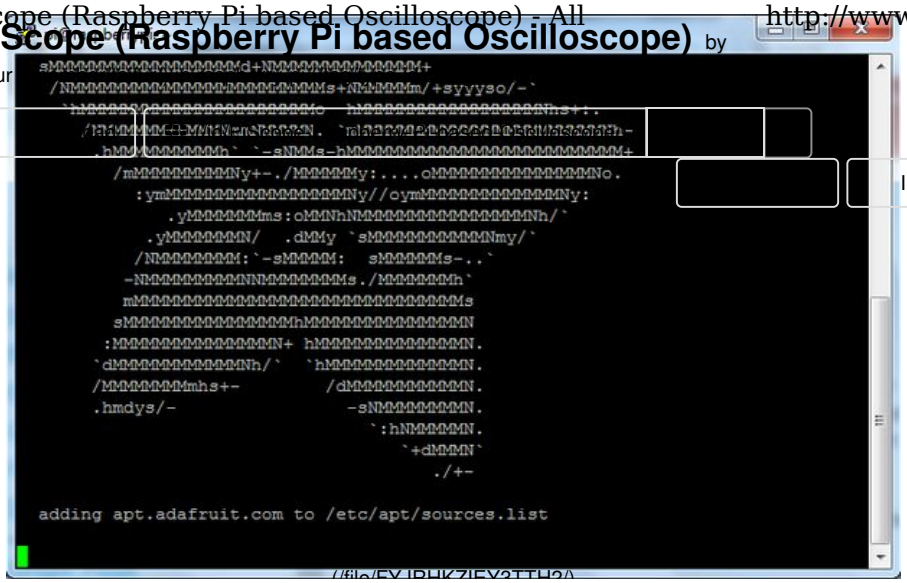
```
<code>
set key inside right top
set title "ADC readout"
set xlabel "Time [ns]"
set ylabel "Voltage [V]"
plot "data. txt" using 1: 2 title ' ADC1' with lines, \
"data. txt" using 1: 3 title ' ADC2' with lines
```

More information on gnuplot can be found at: http://www.gnuplot.info (http://www.gnuplot.info)/

readout.cpp (https://cdn.instructables.com/ORIG/FJP/XR8A/IEY3TSI4/FJPXR8AIEY3TSI4.cpp)

Step 10: Making the thing portable

PiScope (Raspberry Pi based Oscilloscope) by taifur



I Made it! Favorite Share

Show All Items

If you want to make the PiScope portable you can use pi TFT for displaying the data. For that you need to prepare your Pi for TFT. I used adafruit 2.8 inch TFT.

To download and run it, simply run the following commands:

```
curl -SLs https://apt.adafruit.com/add-pin | sudo bash
```

```
sudo apt-get install raspberrypi-bootloader
```

```
sudo apt-get install adafruit-pitft-helper
```

Now type:

```
sudo adafruit-pitft-helper -t 28r
```

At the end you will be prompted on whether you want the text console to appear on the PiTFT. Answer N . Thats it!

Run sudo reboot to try out your fancy new PiTFT.

More details of Pi TFT (https://learn.adafruit.com/adafruit-pitft-28-inch-resistive-touchscreen-display-raspberry-pi/assembly)

Now, plot your ADC data using gnuploat and enjoy your PiScope.

All related file is attached in a single zip file, you can download it.

Download (https://cdn.instructables.com/ORIG/FZ9/G4JY/IEY3TW2S/FZ9G4JYIEY3TW2S.zip)

PiScope (Raspberry Pi based Oscilloscope)

by taifur (/member/taifur/) in raspberry-pi (/tag/type-id/category-technology/channel-raspberry-pi/)

PiScope.zip (https://cdn.instructables.com/ORIG/FZ9/G4JY/IEY3TW2S/FZ9G4JYIEY3TW2S.zip)

Download (/id/PiScope-Raspberry-Pi-based-Oscilloscope/) 10 Steps

adafruit-pi-tft.pdf (https://cdn.instructables.com/ORIG/F3W/5J9N/IF3SSYTJ/F3W5J9NIF3SSYTJ.pdf)

Download (https://cdn.instructables.com/ORIG/F3W/5J9N/IF3SSYTJ/F3W5J9NIF3SSYTJ.pdf)

+ Collection | I Made it! | Favorite | Share

We have a be nice comment policy.
Please be positive and constructive.

I Made it! | Add Images | Post Comment

ReedB9 (/member/ReedB9) 19 days ago Reply

Is there a replacement chip for the CA3306, which is no longer available?

Nelvig (/member/Nelvig) ▸ ReedB9 (/member/ReedB9) 10 days ago Reply

1PCS CA3306E 6-Bit, 15 MSPS, Flash A/D Converters DIP18 (http://www.ebay.com/itm/322190444602)

Here you are!

GaryB180 (/member/GaryB180) 4 months ago Reply

A truly outstanding piece of work!

I had to tweak the
piscript.sh slightly: after extracting the source my folder was called simply "linux" and not "raspberrypi-linux", and I had no /proc/config.gz so had to run the command "sudo modprobe configs"

chartog (/member/chartog) ▸ GaryB180 (/member/GaryB180) a month ago Reply

Hello Gary,

What exactly did you do differently from the provided commands? Where did you run "sudo modprobe configs"?

bdeutl (/member/bdeutl) 2 months ago Reply

This is one excellent write up! Thank you for posting your work.

PiScope (Raspberry Pi based Oscilloscope)

by taifur (/member/taifur/) in raspberry pi (/tag/type-id/category-technology/channel-raspberry-pi/)

I am in the middle of building this and have completed all of the kernel module work and the python program as well. All seems to be working well. I just have a question about the level shifter being used. It appears as though the 5v and 3.3 v are swapped(which makes sense for protecting the pi), my question is the level shifter capable of this swap without damaging it? Also, what is the double line symbol that is between the ground for the supply voltage to the level shifter?

Thank you again for this write up!

bdeutl (/member/bdeutl) ▸ bdeutl (/member/bdeutl) 2 months ago Reply

Never mind, it's a bidirectional level shifter, I will just swap the inputs altogether.

AAtusen (/member/AAtusen) 5 months ago Reply

Nice writeup! Thank you for sharing! :D

PeterB120 (/member/PeterB120) 10 months ago Reply

hello. I am using a raspberry pi 2 and I cant compile Scope-drv.ko. it says module not found. please assist me? thanks