Control of Remote Control Mains Sockets – By Geoff Johnson.

This article covers an easy to build, cheap, and above all, safe way to control mains powered devices with a Raspberry PI. Nothing in this project involves going anywhere near any dangerous voltages. Soldering is limited to just a few joints, and source code can be downloaded for the software part.

THE STORY.

I’m sure I’m not the only person who jumped at the chance to buy a machine like the Raspberry Pi - then wondered what they’d do with it. It isn’t the first time I’ve bought something that seemed like a good idea, then been stuck for a use for it. A couple of years ago, I noticed a sign on a petrol pump advertising remote controlled power sockets at £5 for a pack of 3. So, I bought them, tried them out with a few random appliances, stuck them in the cupboard and forgot about them.

This brings me to the idea of putting these two compulsive purchases together.

A lot of this was made up as I went along, rather than designed from the outset, so I’ll follow the development chronologically, rather than describe the system fully in one go.

The first step was to identify the signalling used by the sockets so I could try to mimic it using the Raspberry Pi. The communication between the remote and the sockets is radio based, allowing control without line of sight. This allows sockets in various rooms to be controlled from one location. There’s a sticker on the back of the sockets saying 433.92MHz, so I searched ebay for “433MHz receiver” to find a suitable receiver. This found me a transmitter and receiver for only £1.60 including Postage and Packing. It’s shipped from China, so there’s a fairly high chance of it going missing in the post, but I’ve always found professional Far-East sellers will happily send another if that happens.

I had to solder an antenna wire onto each one, but it’s not a difficult piece of soldering, though the pads are quite small.

To read the code from the remote control, I connected the receiver module to a computer mic socket. I have a 5V power supply lying around, but there’s a 5V output on the Raspberry Pi that would serve the purpose. The output from the module is 5V digital, which isn’t suitable to be connected straight into a computer audio socket, so I connected the output of the module via a 1MOhm resistor to the mic socket of my laptop, borrowing the cable from my speakers to make the connection. I assembled the circuit on breadboard, but I’ll describe an alternative later.

Audacity (excellent freeware, just Google for it) is great software for examining signals like this. Once I was satisfied that the recording level was about right, I started recording and pressed one of the buttons on the remote. Having stopped the recording I was able to zoom in on the area where the signal from the remote was. The signal repeated over and over until the remote button is released. Short pulses in it seemed to be about 0.25 ms and the long ones 3 times as long. So I could convert it to this binary string.  
111110001000100010001000100010001000101110001011100010001011100010001011101110111011101110001000100010111111

This is make or break time for the project. If you have come this far and cannot find a repeating pattern, that appears when you press the remote control button, your sockets may not be using the simple AM signalling that this project relies on.

To be able to do anything with the captured data, I needed to wire the transmitter up. But first, I powered it from the 5V supply and checked the input pin with a volt meter. On some devices, the supply voltage is present on the input pins when there’s little or no load. This could potentially damage the Raspberry Pi, so I wanted to make sure. There was no voltage present, so I went ahead and wired it up. The transmitter is intended to stay connected to the Raspberry PI, so it’s powered from the +5V pin of the GPIO. I connected GPIO 7 of the Raspberry PI to the data pin and the GND of the transmitter to the GND of the Raspberry GPIO connector. The 3.3V signals are enough to drive the transmitter, though I only found this out by trying it.

From the waveform in audacity, I could see what I would need to send, and how fast I would need to send it. Linux is a multi tasking OS and something else could use the CPU at just the wrong time, which would leave my program outputting bits at the wrong time, so I repeat the output 10 times hoping that at least one gets transmitted correctly. My initial rough cut of the software was based on the first GPIO example in C from the RPi Low-level peripherals page on the elinux.org page. Optimistically, I plugged the RC socket in for the test run. When I ran my software on the Raspberry Pi, and captured the signal with the laptop, receiver and audacity, I could see that the waveform was about right, but upside down. Needless to say, the socket didn’t do anything. I flipped all the bits in my output stream, and re-ran the test code. This time the socket switched on, which surprised me a bit as I expected to have to do a lot more tweaking. The waveform in Audacity matched the one I’d captured earlier pretty well too. This just left me with the other buttons to transcribe.

DETAILS.

So, that’s the story, now for a bit more detail on the hardware and software.

HARDWARE.

The connector for the GPIO and the cable I used is from an old PC floppy disk drive cable. These have a wider cable and plug than the Raspberry Pi GPIO connector, but you can fit the plug onto the GPIO pins with part of the plug going off the end. Of course, this won’t work if your Raspberry Pi is in a case. One end of the cables has the ribbon split into three, with the middle one of the three twisted over. I used this end of the cable as it left the cable pointing out away from the Raspberry Pi. The other end of my cable would have gone over the top of it. With the cable this way around there are 8 wires in the ribbon cable off the end of the GPIO connector. I cut these off quite close to the connector and used some of the wire for the antenna wires mentioned earlier. Then there is a single wire connected to the GPIO pins before the first split in the cable, this wire isn’t used for the transmitter though. The first wire after the split is +5V, I use this to power the transmitter. After this are 3 other wires we don’t need. Then comes the GND connection. Finally, right at the bottom of the ribbon is GPIO7, the pin I use to control the transmitter. The +5V and GND lines It could also be used to power the receiver while collecting the data from the remote control.

On the other end of my ribbon cable is a very small piece of stripboard. Soldered to this board is a connector for the transmitter, made by cutting down an IC socket. Another way to connect the transmitter module would be to use the connector from the other end of the floppy drive cable. The transmitter would go into three of the pins. The three wires connecting to the transmitter in that socket could be connected to the +5V, GND and GPIO7 wires from the connector on the Raspberry Pi. A chocolate block type connector would be fine for connecting the wires as the signals aren’t very high speed. If your transmitter has another pin for the antenna connection, you would just need to trim the wire on that pin to the correct length and do no soldering at all.

SOFTWARE.

As the GPIO in the example I followed directly accessed memory, it needed to be run by the root user. To make my life easy, I developed and tested it while logged in as root. All commands given here assume you’ll be doing the same. Everything is done from the command line, so unless your system is set to jump straight to the graphical front end, just stay at the command line. Personally, I don’t even have a monitor and keyboard connected, I just use ssh to connect over the network. There’s a windows application called putty to connect from a windows PC.

To create a directory in which to keep the source code and the executable file while under development, type the following  
mkdir gpio  
cd gpio

My source code can be downloaded from [www.hoagieshouse.com](http://www.hoagieshouse.com), just follow the links and save the file in the directory you just created. It takes 2 parameters, the channel and on or off. You’ll need to edit the code to replace my remote control codes with your own. My favourite editor in Linux is nano, because it’s relatively easy to use, and is usually installed on most systems. To access the editor type  
nano switch.cpp

Be carefull to retain the quote marks around the codes in the source. When you have changed the codes, quit by pressing CTRL X, answer Y to the question about saving the file and accept the filename switch.cpp.

To build the executable file, type   
g++ -o switch switch.cpp

Test it with your sockets by typing  
./switch 1 on

If it works, you will probably want to be able to run it as any user. Type the following commands to do this.  
chmod +s switch  
mv switch /usr/bin/

To schedule the sockets to come on and off at certain times, you can use something called cron jobs. Just type  
crontab –e

You will be able to edit a file that controls scheduled jobs. The format is described in the file, but to try something out, add these lines to the bottom of the file  
0 \* \* \* \* switch 1 on  
10 \* \* \* \* switch 1 off

This will turn socket one on for the first ten minutes of every hour.

So far, this is not very user friendly. A web based interface would be a lot nicer. My web interface allows the 4 channels to be switched on and off, but I’ve not gone as far as to add any scheduling to it. First, install mini-httpd to act as the web server, to do this, from a shell prompt type  
apt-get install mini-httpd  
  
You can download the files for the web interface from [www.hoagieshouse.com](http://www.hoagieshouse.com). In the zip file is the config file for mini-httpd, and the var/www directory, which is where I put web page and cgi programs. These shouldn’t need any modification, but will need to be copied to the correct locations. The HTML file uses a primitive AJAX request to run the CGI script with the channel and on/off parameters. The CGI script just pulls the parameters from the query and calls the switch program with them.