Lightwave2Gazco

Intro

This document describes a 433MHz gateway device and Raspberry Pi software to allow adding a non LightwaveRF device so that it behaves like a Lightwave device and may be controlled by the LightwaveRF App and also by Amazon Echo/Dot Alexa voice control.

The example here used here was a Gazco Gas fireplace which has a 433Mhz remote control with 4 buttons. The idea was to simulate a basic LightwaveRF switch to perform on / off functions for the fireplace. In particular I can now say ' Alexa: Turn Lounge Fireplace On' instead of fiddling with the remote.

The basic scheme is to listen for LightwaveRF 433MHz commands, pick out ones of interest and then transmit different 433MHz signals to perform the functions required.

The scheme here is extendable to simulate multiple Lightwave Devices using the same hardware and different output protocols. Similarly the output side could be extended to handle different ways of forwarding commands like using IR Blaster software.

Hardware

The hardware consists of a simple peripheral for the Raspberry Pi interfaced into the GPIO with 5 connections (+5V, +3.3V, GND, GPIO pin (24) for 433MHz RXD and GPIO Pin for 433MHz TXD (25).

Three small readily obtained module boards were used.

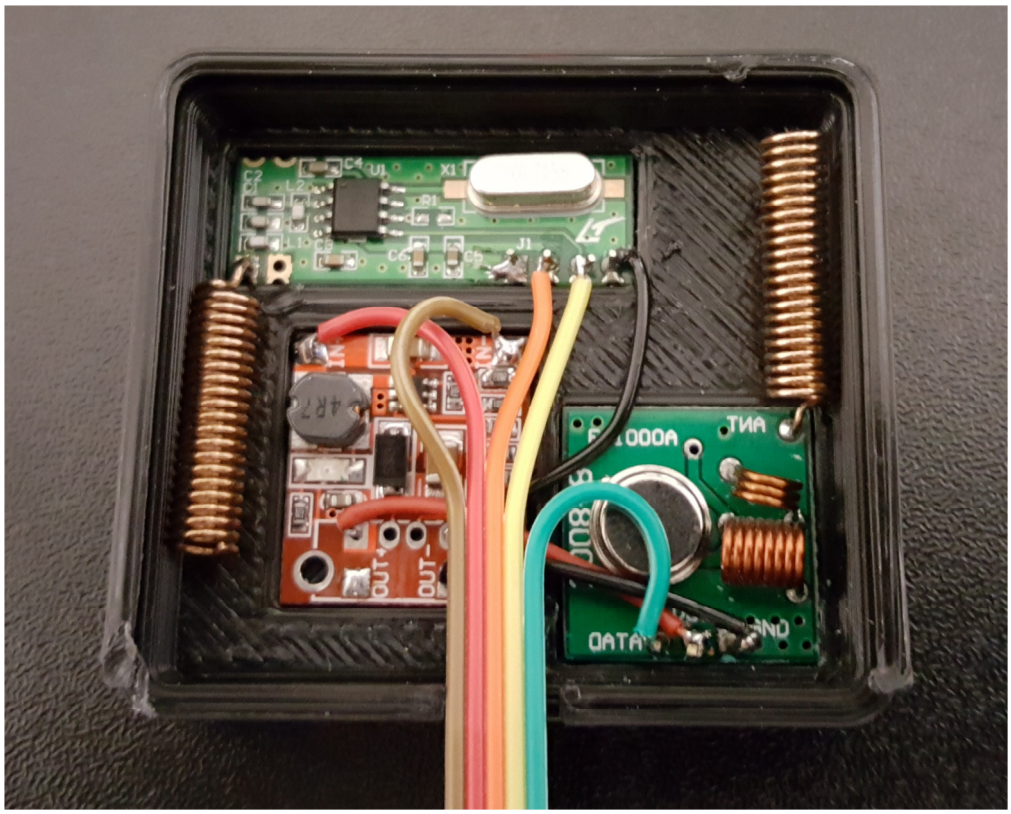
A simple OOK 433MHz receiver. Various modules are available. I prefer to use superheterodyne based ones as they are more reliable over longer ranges.

A simple OOK 433MHz transmitter. The simple modules can be powered from 3 to +12V and have a data pin that can be driven from 3.3V GPIO pins no matter what voltage is used. Higher supply voltages give higher transmit power and longer range.

A simple boost voltage converter. This is used just to step up the +5V supply to +10V to supply the transmitter and give it more range.

I put the 3 modules in a 3D printed box to give a neat finish. The ribbon cable just plugs into the Raspberry Pi GPIO.

The wiring is very simple and can be got from the picture of the box below.



The receive and transmit modules have little antennas added. A sliding lid finishes off the box.

Software

The software is based on my Raspberry Pi LightwaveRF libraries using the custom extended Pigpio method. This gives very low overheads and mean the software can be run on any spare Raspberry. Details of the LightwaveRF library support is available at
https://github.com/roberttidey/LightwaveRF

The library here has been extended to allow different protocols to be used for transmission and in this case a Gazco protocol has been added.

The main routine is a simple python program which uses these libraries to wait for LightwaveRF messages to be received, matches them against expected values and then issues Gazco commands if appropriate.

No explicit pairing is required as the messages being decoded are stored in the code.

The program is run by first starting the pigpio daemon (sudo pigpiod) and then running the python program (sudo python lwrf2Gazco.py) The python will run forever until terminated by a CTRL-C. It amy also be run in the background (sudo python lwrf2Gazco.py &).

Installation

The following steps need to be done for the base install.

1. Install Pigpio using the instructions available at the pigpio site. Make and install the standard Pigpio but do not run the demon yet.

2. Retrieve the files from my repository. Copy the 2 files from the c-custom folder into the PIGPIO folder and make and install again
3. Copy the lwrf2Gazco.py file into ~/python folder

Configuration

The lwrf2Gazco.py file needs to be edited in the following ways.

1. Change the GPIO pins to match the hardware connection if not using GPIO pins 24 and 25
2. Edit the expected LightwaveRF commands for the device on/off to match your environment. The distributed file does not contain working data. See below for method to do find appropriate values.
3. Edit the Gazco commands to be used for the device on/off to match your environment. The distributed file does not contain working data. See below for method to do find appropriate values.

Finding LightwaveRF on/off commands

First add a new device into the LightwaverRF App to match what you want. E.g. Add Fireplace into Lounge room.

Run the lwrfDevice program (sudo python lwrfDevice.py) This will run for about a minute and print out any Lightwaverf messages it receives. Use the lightwave app to send an on and an off for the new device. These will print out. Note them down and use these values in lwrf2Gazco.py

Gazco protocol and finding on/off command

The Gazco protocol is very simple. A '1' is encoded as 580 uSec on pulse followed by a 320 uSec off period. A '0' is encoded as a 320 uSec on pulse followed by a 580 uSec off period. A message consists of a 'training' start pulse (500 uSec on 400 uSec off) and then 21 bits of 1's and 0's. I think the first 16 bits are a unique address and the remaining 5 are the commands.

I don't have a program to read Gazco data programmatically at the moment and I used a general 433MHz sniffer technique to capture the raw pulse data and then worked out the commands for my particular fireplace corresponding to on and off. An alternative method would be to use a logic analyser or the pigpio piscope to capture the data. Although there can be a lot of background noise it is normally easy to see the message data as the noise goes away and the message can be seen to be repeated.

For sniffing I used rxrf.py (sudo python rfrx.py). This runs for a few seconds and produces a couple of text files where it logs the state of the received data 0/1 and the delay in microseconds from the previous transition. A normal give-away for a set of messages is the presence of an entry showing a long period of '0'. This is the quiet period between repeats of the same message after the receiver has adjusted its gain. The message bits are then those between 2 inter-message gaps of '0'