iLumos Light Switch protocol

Introduction

This document describes the 433MHz transmission system used to control the iLumos range of light switches,; both dimmer and on / off.

This applies to those switches that may be controlled by the little 433Mhz remote controls. They are also said to be usable with the Broadlink RM Pro IR / RF wifi interface units but in practice this can be quite problematic as the training method used by the Broadlink unit does not seem to capture the protocol messages very well. Even when 'trained' they do not operate very reliably (as tired in Dec 2018).

The messages from the remotes was studied using a 433Mhz receiver interface to a Raspberry Pi. Firstly the raw transmission data was captured to see what was being transmitted. From this it proved possible to ascertain the basic bit timing and the message contents. This allowed a specific capture routine to be used to decode the message contents form a range of different remotes.

The basic structure is reasonably clear and certainly allows for a separate transmitter to control the switches via a 433Mhz transmitter. How the commands are logically constructed is not fully understood at the moment.

A wifi operated remote control unit based on an ESP8266 module, with a 433MHz transmitter has been constructed and successfully controls the switches. It also integrates an IR transmitter so that the same unit can be used to control IR based devices like televisions.

Basic bit timing and code structure

The iLumos uses 3 types of pulse sequence to construct a message.

Header Start

Data One

Data Zero

The pulses are a little bit shorter than many comparable 433MHz devices so it is important to have reasonably fast receiver to capture these accurately. Some of the superhet type devices are lower bandwidth and distort the timings a bit.

A code consists of a header start followed by 24 data bits giving a sequence consisting of 50 on/off pulses and a total duration of about 13 msec.

The codes are repeated about 44 times for a single remote control button press. These are sent back to back with no gaps. I.e the last data bit of a message is immediately followed by the next header start. The total duration for a button press is about 500mSec.

As there is no pre-amble the first one or two code sequences may be received with some timing errors as the automatic gain control in the receiver settles down. The number of repeats though ensures a good reception providing a good transmitter and aerial is used.

The lack of pre-amble and the tight timing may be contributing to the training difficulties in the Broadcom unit.

Message structure

The 24 bits code appears to be arranged in 2 parts. The first 16 bits seem to be an identifier for the particular remote and it is probably this part the switch is primarily paired against, so that it then recognises and accepts commands if this part matches.

The remaining 8 bits are probably the command. The remotes use 1 button to control an on / off switch (On/Off Toggle), and 3 buttons to control a dimmer (On/Off toggle). A fourth button can be used as an AllOff switch. The simple key ring remote has just the 4 buttons. The multibutton remote has buttons for up to 3 dimmers, 6 on/off switches and an allOff button.

There appears to be a number of command codes that can be utilised as the on/off toggle and once a switch is paired with the address and this command code thaen that is what it uses. Six triplets of command code have been identified for dimmer use, plus 6 on/off toggle codes for switches. This was from the 4 mini remotes and 1 full remote available. There may be more and it is easy to capture these using the utility described later. Please inform the author if new ones are discovered.

The hex codes captured are shown below where aaaa represents the 4 hex digit id of the remote in use.

	On/Off	Down	Up
Dimmer	aaaa8C	aaaaAE	aaaa9D
Dimmer	aaaa9C	aaaaBE	aaaaAD
Dimmer	aaaaFC	aaaa1E	aaaa0D
Dimmer	aaaaAC	aaaaDE	aaaaCD
Dimmer	aaaa04	aaaa6A	aaaa59
Dimmer	aaaa37	aaaaD2	aaaaE3
Switch	aaaaBF		
Switch	aaaa15		
Switch	aaaaB0		
Switch	aaaa48		
Switch	aaaa26		
Switch	aaaaC1		
AllOff	aaaa7B		
AllOff	aaaa8B		
AllOff	aaaaEB		
AllOff	aaaa9B		

Transmission Unit

A transmitter based on a previous ESP8266 IR controller was used to transmit the data. This has a bit message and bitTx library which can handle both modulated IR transmissions and the pulse data required for the rf transmitters. This allows the device to be used for both iLumos switch control and IR devices like TVs. Other definitions can be added for both other RF or IR devices by adding text config files to the unit.

Capture utility

First stage capture was done using a raw pulse capture program (rxrf.py) written in python for a Raspberry Pi. A 433MHz receiver was coupled into GPIO24. Note that a level shifter or potential divider may be needed if the RX puts out 5V data. Run it by 'sudp python rxrf.py'. This utility just writes 2 text files (rxdara1 and rxdata2) containing the pulse data in terms of levels and durations in uSec. These files can be examined to discover the location of transmissions, the bit timings, and the basic message structure. If the remote is held close to the receiver the agc circuit in the receiver normally means there is a long quiet period on the data output at the end of a message before the gain is ramped up. This makes it easier to find the message ends in the raw data amongst the background noise. The preceding data can then be seen more easily analysed.

Once the basic structure was known a second stage capture program (iLumosrf.py) was used to capture the codes associated with each button press. It looks for the start pulses and then checks to

see if the message structure is correct. It skips over the first few messages to allow the agc of the receiver to settle. This may be used to find the addresses used by other iLumos remote controls. Run ‘sudo python ilumosrf.py’. It waits for a key press then puts out the hex code for this. Use CTRL-C to terminate capturing codes.