

# Radiator Remote Control Analysis

## General

Remote control is a simple 6 button infra red remote control. Each button when pressed issues a repeating identical message all the time the control is held down.

The 6 buttons are

- On/Off – Toggles radiator on/off. On moves into Standby mode
- Temp – Moves into Temp setting. Then use up /down
- Mode – Cycles round Standby, Frost Free, Eco, Comfortable
- Time – Moves into Time setting. Up/Down sets a countdown timer which turns on from Standby, or turns off from the other modes
- Down
- Up



## Messages

Messages are 12 bits long

Bits are just encoded as a simple on/off pulse. High frequency modulation is used to modulate the IR on at about 37KHz. The rest of the description here just uses the logic levels but where a IR-On is shown it is actually a square wave at about 37KHz.

Each bit is encoded as a pulse 1.7mS long.

A 0 is 1.3mS IR-On, followed by 0.4mS IR-Off

A 1 is 0.45mS IR-On, followed by 1.25mS IR-Off (Probably periods as 0 but rise time affects)

Message takes  $12 * 1.7\text{mS}$  to send (20.4mS)

Gap between repeats of same message 6.6mS

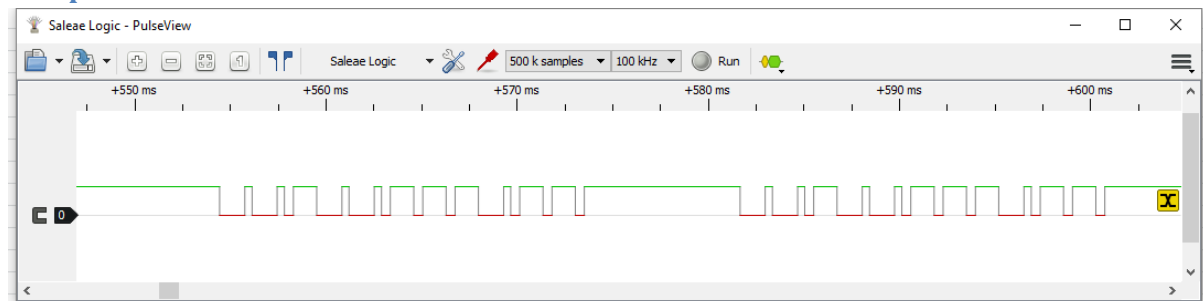
A longer gap would be present when button released (say 500mS). So a sequence of ups for example could be send Up repeats for 250mS, pause 500mS, send Ups for 250mS etc.

## Button Messages

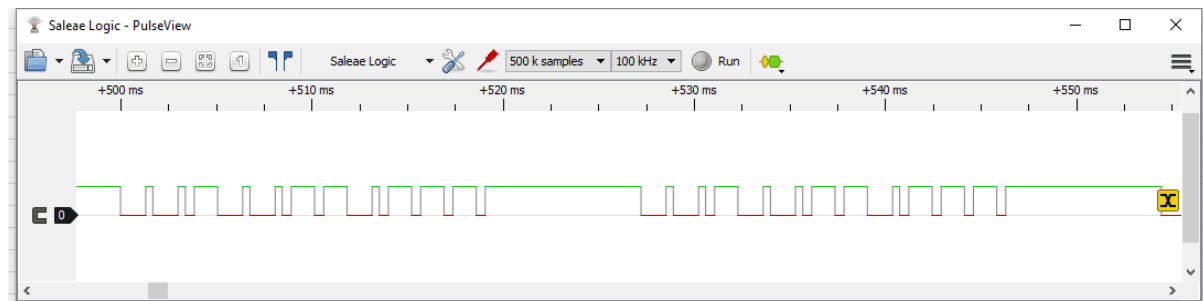
### On/Off



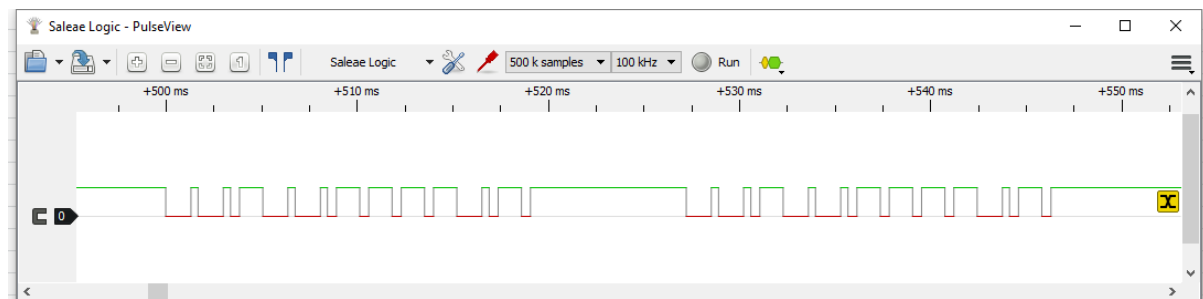
### Temp



### Mode



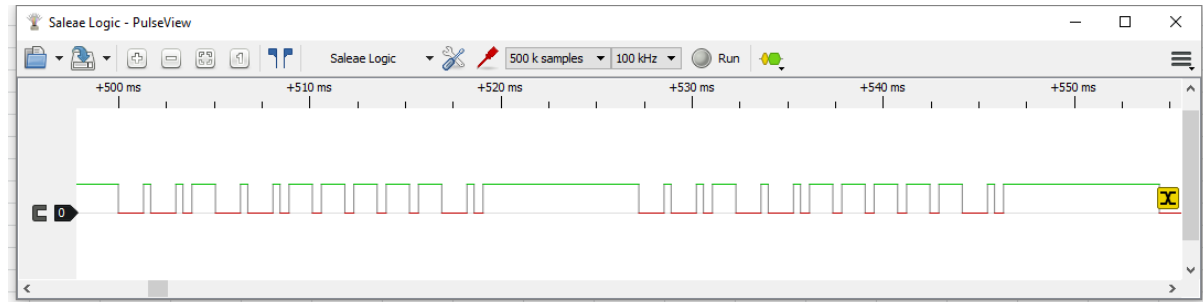
### Time



## Down



## Up



DataBit	0	1	2	3	4	5	6	7	8	9	10	11
On/Off	0	0	1	0	0	1	1	1	1	1	1	0
Temp	0	0	1	0	0	1	1	1	0	1	1	1
Mode	0	0	1	0	0	1	1	0	1	1	1	1
Time	0	0	1	0	0	1	1	1	1	0	1	1
Down	0	0	1	0	0	0	1	1	1	0	0	1
Up	0	0	1	0	0	1	1	1	1	1	0	1

## Control Strategies

Main purpose is to provide proper time of day control rather than simplistic countdown which needs to be set up every time. Strategies are complicated by the toggle nature of on/off and the cyclic mode.

Use Photon with IR LED to simulate remote control. Photon can get time easily from Internet. Phone app could be used to set up and provide manual control using Photon cloud integration. If Phone App was used exclusively instead of remote then it would be easier to know what state the radiator was in as all commands pass through Photon.

### Assume Off Strategy

If radiator is assumed to be off then it can be turned on by sending an On/Off message followed by a sequence of Mode, Mode, Mode messages.

Similarly if it is assumed to be On then a simple On/Off message would turn it off.

### Use timer strategy

Leave radiator permanently in Standby. Turn on by sending a Time, Time sequence (to get it to minutes) followed by up. 1 minute later radiator will turn on.

Similarly if On then send Time, Time, Up and 1 minute later it will turn Off

### Other strategies

Explore sending other code sequences not supported by remote buttons to see if there are explicit off and on commands

Find some way of detecting whether radiator is on (detect status light or temp sensor on radiator)

### Basic Library

Library supports transmission of button press sequences. A sequence is defined as code words (uint16\_t) in a buffer. Code words have 3 types of value

- 0x0000 - 0x1000 defines a button press code (Bit 11 transmitted first)
- 0x1000 – 0x7fff unused
- 0x8000 – 0xffff delay in units of 25mS (mask of top bit)
- 0xffff end of sequence (must be present)

The library is constructed as a state machine inside an interrupt service routine called at 13 uSec intervals while transmission of a sequence is in progress. This fast rate is used to provide the capability to modulate the IR-On signal. Outside of this it is prescaled by 7 so the bulk of the state machine is running at 91 uSec rate.

The API into the library consists of 5 calls

- void radtx\_setup(int pin, byte repeats, byte invert, int period) Called once to define basic transmission configuration
- boolean radtx\_free() Called to see if a transmission is in progress. Should be used to check when it is safe to send another sequence
- void radtx\_send(uint16\_t \*msg) Used to send a message sequence. Automatically activates the service routine to initiate sending
- void rad\_debug() Used during debug. Returns an int (default 0) radtx\_debug which can be set in various routines
- void radtx\_update(byte repeats, int uSec) Allows tweaking base timing and code repeats

Support routines consist of variants of timer handlers to match the sbc board type. Which set is used is determined by #define in RadTx.h

- extern void rad\_timer\_Setup(void (\*isrCallback)(), int period)
- extern void rad\_timer\_SetPeriod(int period)
- extern void rad\_timer\_Start()
- extern void rad\_timer\_Stop()

Note that Particle versions require use of SparkIntervalTimer library to control timer.

### Basic Run Time

A simple Particle (Photon) run time app RadTXTTest.ino uses the library.

It defines 9 message sequences of 10 button presses. It sets up a function called sendMsg which may be called via the Particle Cloud with

```
https://api.particle.io/v1/devices/devicecode/sendMsg?access_token=accesstoken"
method="POST"
```

where there is a named POST variable args containing a string value 0 to 8 which selects one of the pre-defined messages sequences.

If the arg string starts with a \$ then the rest of the string is decoded as a raw code to send out.

The remote buttons have predefined constants

BTN_ONOFF	0x027e
BTN_TEMP	0x0277
BTN_MODE	0x026f
BTN_TIME	0x027b
BTN_DOWN	0x0239
BTN_UP	0x027d

Timing diagrams here just show the logic levels without modulation.

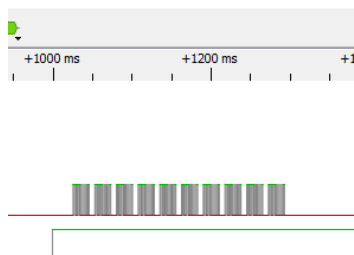
Example output for a message sequence

BTN\_ONOFF,BTN\_MODE,BTN\_MODE,BTN\_MODE,0x8078,BTN\_ONOFF,0x8028,0xffff,0,0

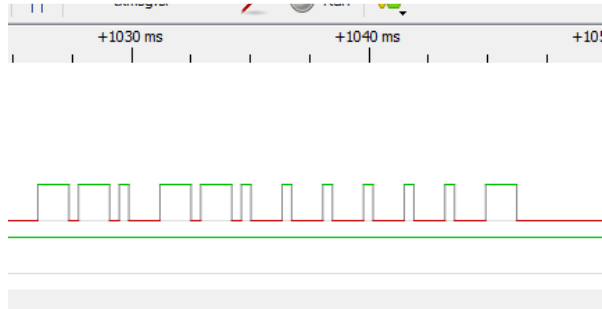
### Overall Sequence transmit



### One Button press repeating 10 times



## One button timing



It also supports reading the temperature from a OneWire DS18B20 sensor. OneWire and Dallas are utilized for this. A `#define DS18B20` at the top of the app code determines whether this is included.

## Schedule functions within the app

The App supports a schedule consisting of 4 on/off times per day for each day of the week (0=Sunday). On/Off values are in minutes (0-1439) and values > 2047 are events which are disabled. A HolidayMode (0/1) disables the whole schedule to Off when set.

A particle variable called schedule returns a 56 value csv string which the values are the 4 on/off values for the 7 days of the week. E.g. first 8 values are on/off minute values for Sunday.

A particle variable called status returns a 6 value csv string. radstate (0/1), lastEventDay (0-6), lastEventTime interval (0-7), current time, temperature, debug string.

A particle variable called config returns a 5 value csv string. HolidayMode (0/1), DaylightSavingType (0-2), timeZoneOffsetHours, TimingPeriod (uSec), codeRepeats. TiminPeriod sets base period for transmissions (default 13 uSec) and codeRepeats sets number of repeats of each code transmission (default 10).

A particle function called receiveSch(newSch) allows the schedule for 1 day of the week to be updated. It is a 9 value csv string consisting of the day number and the 4 on/off times.

A particle function called receiveConf(newConf) allows the config to be updated. It is a 5 value csv string of the same structure as the 'conf' string above.

A particle function called sendMsg(msg) allows sending one of the predefined commands 0-8. If 'msg' starts with a \$ then the rest of the message is interpreted as a raw transmit code and is sent using preset 9.

A particle function called receiveConf(newConf) allows for updating config integer values. They are sent as a csv. Currently 3 values are used zone (time offset), dstType (daylight saving mode 0=European, 1= USA, 2 = Off), holidayMode (if non zero the radiator is off all the time).

## Web site

A web site (under development) to control the app from a browser is contained in the www folder. It expects to be hosted on a web server supporting php.

Browser looks like

Radiator Remote

Schedule

Settings

● Thu 31 Mar 23:30

On	6:00	⌚	Off	8:00	⌚	Su	Mo	Tu	We	Th	Fr	Sa	✓	🗑️
On	9:00	⌚	Off	11:00	⌚	Su	Mo	Tu	We	Th	Fr	Sa	✓	🗑️
On	13:01	⌚	Off	14:00	⌚	Su	Mo	Tu	We	Th	Fr	Sa	✓	🗑️
On	18:00	⌚	Off	23:50	⌚	Su	Mo	Tu	We	Th	Fr	Sa	✓	🗑️
On	7:00	⌚	Off	9:00	⌚	Su	Mo	Tu	We	Th	Fr	Sa	✓	🗑️
On	10:00	⌚	Off	12:00	⌚	Su	Mo	Tu	We	Th	Fr	Sa	✓	🗑️
On	18:00	⌚	Off	23:33	⌚	Su	Mo	Tu	We	Th	Fr	Sa	✓	🗑️
On	15:15	⌚	Off	23:00	⌚	Su	Mo	Tu	We	Th	Fr	Sa	✓	🗑️
On	23:41	⌚	Off	23:42	⌚	Su	Mo	Tu	We	Th	Fr	Sa	✓	🗑️
On	13:00	⌚	Off	16:00	⌚	Su	Mo	Tu	We	Th	Fr	Sa	✓	🗑️

+ Add

💾 Save

⏻ 🔥 ⚙️ ⌚ ▼ ▲

Code

📶

Each row specifies an on/off period and may be assigned to multiple days of the week by clicking on the day buttons. Rows may also be enabled / disabled to allow them to be temporarily removed from the scheduling.

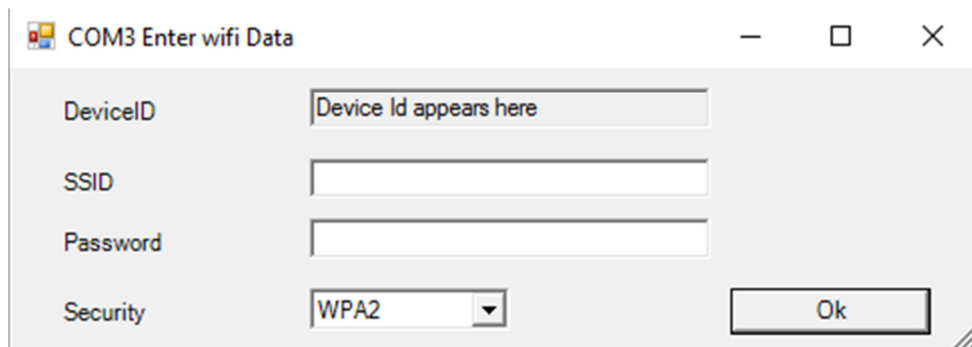
Rows may be deleted and new rows added. Only when Save is pushed are changes committed to the schedule.

The row of buttons at the bottom is the equivalent of the remote control.

## Wifi Set up

Cores or Photons need their wifi set up before they can connect to the cloud. There are various techniques for this including softAP which unfortunately doesn't work on Core only Photon.

An alternative is supported by the wifiSetup.ps1 Powershell script. It is run from a Windows PC with the device USB connected and in flashing blue listening mode (hold mode down for a few seconds). As Powershell scripts are not necessarily enabled a wifiSetup.cmd is provided to enable and run the script. Double click the cmd and you get a simple set up dialog.



COM3 Enter wifi Data

DeviceID: Device Id appears here

SSID:

Password:

Security: WPA2

Ok

The Device ID is shown and may be copy / pasted for use elsewhere.

Enter local wifi SSID, password and security type. Press OK and the device will be set up. It should connect automatically at the end of this.