IR Remote Control Primer

Introduction

IR Remote Controls can send and receive data encoded in various fashions as pulses of infrared light. The various encoding methods that most IR remotes support are grouped under the general term *'Consumer IR'* or CIR.

CIR is generally used to control consumer products such as TVs, DVD players or game consoles with a wireless remote control, but in general can be used for any application that needs to transmit low speed data wirelessly. CIR is a *low speed protocol*. This means its commands generally contain no more then 32-bits of data with a maximum bit rate of 4000 bits/second (but usually much less). There is no concession for anti-collision, so only one code can be transmitting at any time. Transmission distance depends on the power of the transmitter, and the receiver typically needs to be within line of sight. However, the signal can still reach the receiver without line of sight by bouncing off of the walls and ceilings if the transmitter is strong enough. CIR data is transmitted using a modulated bit stream. Data is encoded in the length of the IR light pulses and the spaces between pulses. The pulses of IR light are themselves modulated at a much higher frequency (usually ~38kHz) in order for the receiver to distinguish CIR data from ambient room light.

Data Encoding Properties

CIR codes encode data using a specific encoding scheme. Apart from the actual data part of the bit stream, there are a number of other features to describe. These features can all be specified when transmitting a code using our API, and they are filled in when learning a code.

Header

A CIR code will often start with a header. This is a pulse and space that immediately precedes the data. Usually the pulse is quite long (several ms), and is used by the receiver to adjust its gain control for the strength of the signal.

Trail

The trail is a trailing pulse that follows the data stream but is not part of the data. This is only used in some encodings.

Repeat Code

The repeat code is a series of pulses and spaces that are sent at a specific amount of time after the data packet. This allows the receiver to the know that the button is being held down. Not all encodings have a specific repeatpacket, but all encodings support repetition (either with a repeat code, or simply by repeating the data over and over).

Bit Length

This is the number of bits in the data. Most codes are between 8 and 32 bits long.

Code Length

Codes can have either a constant length or a variable length. Constant length codes will have a specific amount of time within which the code can be sent, which includes the code itself followed by a gap time. The gap time plus the data time adds up to the constant code time, so that if the data takes a bit longer because there are lots of 1's then the gap will be shorter, and if the data time is short because a repeat code is being sent, then the gap time will be much longer. Variable length codes will have a varying data encoding time followed by a constant gap time.

Gap

Gap refers to the time between codes. This is a long (>20ms) space time between codes that helps the receiver see the beginning of the next code. If the code length is constant, this gap time refers to the entire code length - the data time plus the gap time. If the code length is variable, this gap time refers to the constant gap time only.

Minimum Repeat

Some encodings require a code to be repeated a number of times before it is recognized. This usually cannot be automatically detected.

Toggle Mask

CIR data is sometimes toggled for two reasons. In one case some of the data is toggled every time it is sent. This can help the receiver to recognize the data as valid, and is often combined with the Minimum Repeat feature. This can sometimes be detected automatically.

In a more common variant, one bit of data is toggled every time the button is released - in this way when a button is pressed and held down, you get the same code repeated over and over - once the button is released and pressed again, you get the same thing, but with one bit toggled. This helps the receiver to know when a code is being repeated. This 'toggle on repeat' cannot be detected automatically.

Carrier Frequency

The carrier frequency is the frequency used to modulate the IR pulses. IR receivers are tuned to a specific frequency so it's best to transmit on the frequency that they are tuned to receive. Most consumer IR devices use 38kHz, and this is usually set as the default. Others that are sometimes used include 56kHz, 40kHz, 36kHz or 39.2kHz. If you are unsure what frequency your receiver is tuned to, we recommend that you check the datasheet or documentation online.

Duty Cycle

Duty cycle is the period of the carrier frequency. This can be set from 1 to 50, but useful values are in the range 25-50. The default is 50. The duty cycle of an incoming data stream cannot be automatically determined.

Data Encodings

CIR device manufacturers use many different protocols to encode data using pulses of IR light. Few of these protocols have been standardized, and many are very similar but have different bit lengths, or different headers, etc. There are three major ways to encode data that are supported automatically by our devices, and these cover the encoding used by almost all CIR devices:

1. Pulse Distance Modulation (PDM) or Space Encoding

This is the most common encoding scheme. Here, data is encoded by modulating the duration of the space between pulses, while the pulse time remains constant. For example, a ‘1’ could be encoded by a 500us pulse followed by a 1ms space, and a ‘0’ encoded by a 500us pulse followed by a 500us space. The data stream will be terminated by a trailing pulse which lets us determine the length of the last space.

2. Pulse Width Modulation (PWM) or Pulse Encoding

This is like Space encoding, except that the data is encoded by modulation of the pulse width. For example, a pulse of 1.2ms followed by a space of 600us would encode a ‘1’, while a pulse of 600us followed by a space of 600us would encode a ‘0’. There is no need for a trailing pulse because the space length is fixed, and the trailing space can be inferred. This protocol is used by Sony.

3. Bi-Phase or Manchester Encoding

This encoding breaks each bit in the data stream into a fixed length of time, where each time slice is half pulse andhalf space. For example, if the bit time is 800us, a ‘1’ could be encoded by a 400us pulse followed by a 400us space, while a ‘0’ would be encoded by a 400us space followed by a 400us pulse. The PhidgetIR supports two specific variations of Bi-Phase coding, called RC5 and RC6. There are protocols that have been somewhat standardized - with specific bit times and headers, and they can be automatically recognized and used for transmission. There are many other ways to encode data, but they are not covered here, and not supported automatically by our devices. They can be reproduced by capturing / retransmitting a RAW bit stream. One such protocol would be RCMM.

Also note that, with the exception of RC5 and RC6, there is no way of knowing how ‘1’s and ‘0’s were originally encoded by the manufacturer, and the automatic decoding logic used by our IR Remote Controls will simply guess. Therefore, a code returned by the device may be the complement of the same code published elsewhere. That being said, it will always consistently return the same code for a specific bit stream.

Receiving Data

An IR transceiver continuously receives IR data when it is not transmitting. There are three ways to intercept this data.

1. Code data: Use this method if you are wanting to act on specific codes (button presses) but don’t care about reproducing them. The transceiver does its best to decode every code that comes in and displays it as a hexadecimal string (array of bytes). This string should be unique at least on one remote. Access this data using the Code event or the LastCode property.

2. Learn Code data: Use this method if you want to learn or retransmit a code, the transceiver will determine as much about a code as possible and pass the CodeInfo structure/object which contains the code semantics, along with the code string (byte array) itself. This can be used to retransmit the code as it was received. In order to learn a code the button must be held down longer (~1 second). Access this data using the Learn event or the LastLearnedCode property.

3. Raw data: Use this method if the transceiver cannot decode the data automatically, you can access the raw data stream directly. This is an integer array of data in micro-seconds. Each access to the Raw Data array will return an even number of array elements, with the first element always being a space and the last element always being a pulse. Access this data using the RawData event or the getRawData (readRaw) function.

Transmitting Data

Data can be transmitted in two ways. When data is being transmitted, reception is momentarily paused.

1. Code data: Use this method if you want to transmit a code that can be encoded automatically by the PhidgetIR. This will be almost any code. In order to send a code like this, you will need to fill in a CodeInfo structure. The easiest way to get all of these values is to learn the code from the original remote.

Note that the remote will probably use the same CodeInfo settings for every button, with only the actually code (data) changing. You can also fill in the CodeInfo structure manually using known data (for example, in the LIRC remote control database). Just be aware that their codes may not exactly match the codes you need to use because of a most/least significant bit first convention mismatch or the flipped bits methods mentioned above.

2. Raw data: If you can't represent your bit stream using a CodeInfo structure, you can send Raw data directly. This is just an array of pulses and spaces in microseconds. The array must start and end with a pulse. An optional gap can also be specified which serves to guarantee spacing before any more data is transmitted.

Resources

- Database of CIR codes [1]
- General CIR reference with links [2]
- CIR protocols [3]

References

[1] http://www.lirc.org/
[2] http://en.wikipedia.org/wiki/Consumer_IR
[3] http://www.sbprojects.com/projects/ircontrol/ircontrol.htm

Article Sources and Contributors

IR Remote Control Primer *Source*: http://www.phidgets.com/wiki/index.php?title=IR_Remote_Control_Primer *Contributors*: Mparadis