Intel Edison ® IR Communication Fundamentals

Table of Contents

Revision history		
Version	**Date**	**Comment**
1.0	1/26/2016	Initial release

Introduction

To introduce IR communication protocols, it is very helpful to understand the most common usage of IR communications – TV remote controls.

The TV remote control system, fundamentally, is not different than the IR system that we are going to implement in this tutorial. It consists of an IR Receiver circuit, which is one of my circuits on the television itself, and an IR Transmitter circuit, which is an IR LED on the TV remote together with the circuit that determines what information, what signal is sent with each unique button press. The same basic hardware is used in all TV remote controls. In that case, what is the difference between different remote controls? Why does a TV remote control for a Sony CRT TV purchased in 1999 not issue commands to a Panasonic LCD TV purchased in 2008? The difference is only in the data – in the IR communication system.

Actually, the fact that the same basic hardware is used in all remotes is the reason why universal TV remote controls can work. When using a universal TV remote, the first step after inserting the battery is typically referred to as the programming step. If the model of TV being used with the remote is in the book of codes available for that remote, then the remote can just then be programmed with the 3 or 4 digit code corresponding with that model. However, these remotes typically also have a search mode, wherein they will cycle through all the different protocols one by one until one works.

This document is an introduction to the fundamentals of IR communication. This tutorial uses two different Intel Edisons, one connected with the IR Receiver and the other connected to the IR LED. Specific instructions will be provided for the SparkFun IR Receiver Kit.

In this tutorial, you will learn to

1. Understand the basics of IR communication,
2. Transmit and Receive IR Signals using two Intel Edison devices

Things Needed

1. This tutorial assumes the Intel Edison® IR Receiver Shield Kit tutorial was completed, including all of the hardware prepared.

IR Protocols

There are many commercial infrared communication protocols used by companies such as Sony SIRC, Phillips RC5/RC6, and NEC. All these protocols have some sort of preamble which is mapped on to a pattern of HIGHs and LOWs of various durations. Pictured here is a signal received by the IR receiver, taken from the previous tutorial:

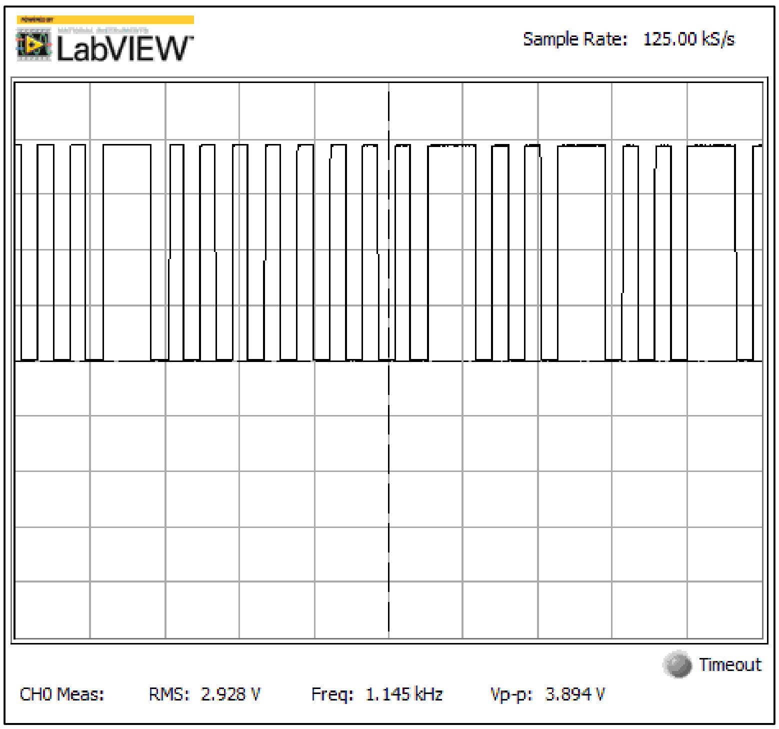

Figure 1 IR Signal Viewed on Oscilloscope

However, just by looking at this signal there is no way to know what it means. In order to interpret this information, we need some outside information: The Phillips TV Code Book:

Insert link to the Phillips Code Book Here.

As you can see, this code book is like 50 pages and we don't want to learn this right away. Instead,

We will use a simple custom IR Communication Protocol that is created to be usable and easy to understand. This protocol can be altered or optimized to fit the needs of any given application.

Before showing our custom protocol, there is one more detail we need to look at a little more closely: PWM and signal modulation.

What is signal modulation?

> *In electronics and telecommunications, modulation is the process of varying one or more properties of a periodic waveform, called the carrier signal, with a modulating signal that typically contains information to be transmitted.*

- Wikipedia

For IR Communication, signals are typically modulated with a frequency between 36-40 kHz. When sending a high value, a constant value is not sent, but instead a period waveform of 36-40 kHz is sent instead.

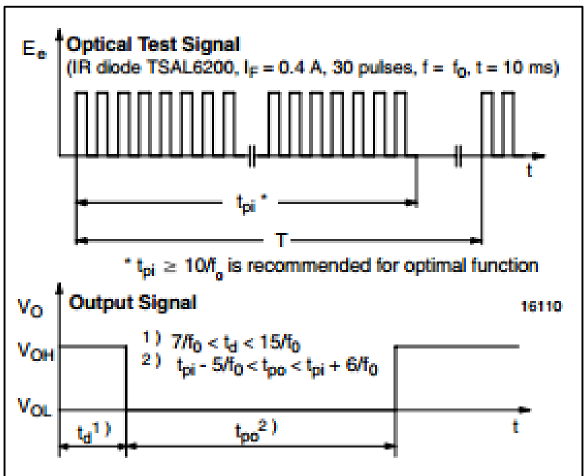

Fig. 1 - Output Active Low

Figure 2 IR Signal Modulation

The IR Receiver diode has a demodulate circuit inside of it that decodes the modulated waveform and translates it into a flat signal. Use of modulation improves robustness to ambient IR noise. Due to the fact only signals at the 36-40 kHz frequency will be accepted, any ambient IR values are ignored.

Custom IR Protocol

All of the details of our protocol are described as follows:

Preamble	2 ms HIGH followed by a 2 ms LOW. Sent (**preamble length**) times.
Bit 0	1 ms HIGH followed by a 5 ms LOW
Bit 1	5 ms HIGH followed by a 1 ms LOW
Preamble Length	**5 (default value)**

Programming Guide

IR Emitter

1. Make sure your hardware is configured as shown in Figure 5.
2. Power on the Edison connected to the IR emitter circuit.
3. Serial connect or SSH into the Edison.
4. **$ git https://github.com/resolutedreamer/IR-Core.git**
5. **$ cd IR-Core**
6. **$ gcc -mraa -o ir_transmit ir_ transmit.c**
7. **$ ./ir_transmit**
8. The emitter will now begin emitting the preamble signal 5 times and then it will send two bits of information, a number from 0 to 3. Because the program was called without passing in any command line arguments, it will send 00 by default. An example of calling it with the argument to send 2 would be

   $ ./ir_transmit 2

   This would transmit the preamble sequence 5 times followed by the pattern for 2 in binary: 10

After the transmitter system is ready, the next step is the receiver.

IR Receiver

1. Power on the Edison connected to the IR receiver.
2. Serial Connect or SSH into the Edison.
3. **$ vi ir_receiver.c**
4. Enter the following C code.

```c
#include <stdio.h>
#include <mraa/gpio.h>

int main() {

    mraa_gpio_context gpio;
    gpio = mraa_gpio_init_raw(48);
    mraa_gpio_dir(gpio, MRAA_GPIO_IN);

    volatile unsigned int received_value;
    while (1) {
        received_value = mraa_gpio_read(gpio);
        printf("%d", received_value);
    }
}
```

Figure 3 ir_receiver.c

5. **$ gcc -lmraa -o ir_ receiver ir_ receiver.c**
6. **$ ./ir_receiver**
7. If the receiver is receiving no signal, the receiver will just display 1's, indicating a high voltage being read.

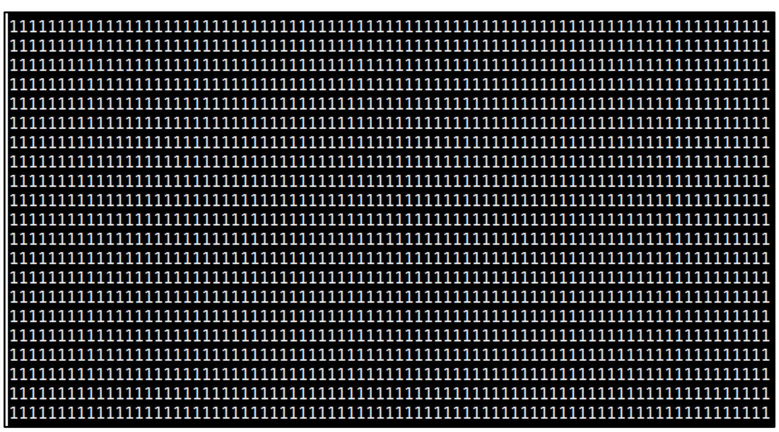

Figure 4 No Signals Received

8. If the receiver is receiving the expected signal from the emitter, the result will be a pattern of 0's and 1's corresponding to when the emitter is emitting high and low values.

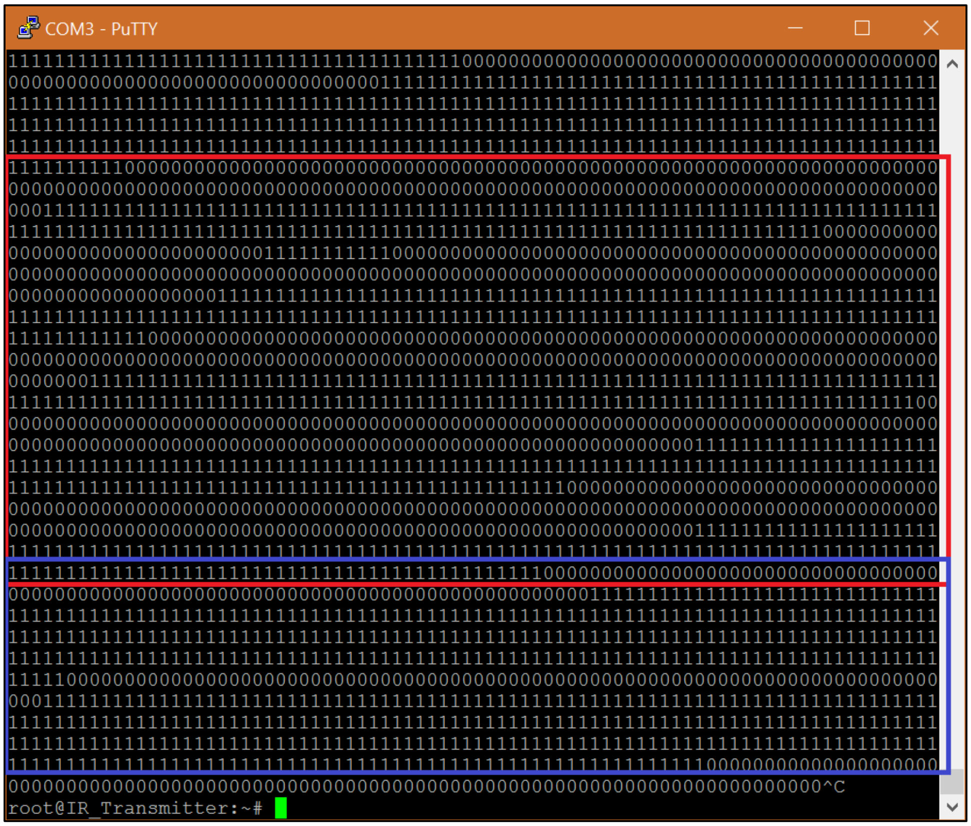

Figure 5 IR Signal Received

In this figure, the portion of the received signal corresponding to the 5 preambles is boxed in red. The portion of the signal corresponding to the 2 low bits (00) is boxed in blue.

Recall from the definition of a low bit above: [1 ms HIGH followed by a 5 ms LOW]. On the transmitter side, HIGH is sent and then LOW is sent.

But the receiver diode is an active low device, so the voltage reading shows LOW followed by HIGH, which is the inverse.

This code is set to sample at the fastest rate that Intel Edison can run the mraa_gpio_read function. However, the ratio of 0’s to 1’s is not as precise as it should be. Instead, replace the code in ir_receiver.c with the following code. This revised version introduces a sampling rate that will take 10 measurements of the pin value per millisecond. Under this code, if the pin is HIGH for 1 milisecond, there should be approximately 10 1’s in a row. The SAMPLING_RATE declaration at the top, 6000, controls the number of times the busy-wait for loop runs to set the sampling timing. The TOTAL_SAMPLES declaration tells it to sample 1000 samples before printing out the values; printf is a very slow function and also affects the sampling rate.

```c
#include <stdio.h>
#include <mraa/gpio.h>

#define SAMPLING_RATE 6000
#define TOTAL_SAMPLES 1000

int main() {

    mraa_gpio_context gpio;
    gpio = mraa_gpio_init_raw(48);
    mraa_gpio_dir(gpio, MRAA_GPIO_IN);

    volatile unsigned int received_value;
    while (1) {
        //received_value = mraa_gpio_read(gpio);
        //printf("%d", received_value);

        int i = 0, j = 0;
        int raw_data[TOTAL_SAMPLES] ;
        unsigned int samples_remaining = TOTAL_SAMPLES;
        i = 0;
        while(samples_remaining > 0 ) {
            raw_data[i] = mraa_gpio_read(gpio);
            for (j = 0 ; j < SAMPLING_RATE ; j++);
            i++;
            samples_remaining--;
        }
        i = 0;

        printf("\nSample Collection Complete\n");

        printf("Raw Received Value\n");
        for (i = 0; i < TOTAL_SAMPLES; i++) {
            printf("%d", raw_data[i]);
        }
    }
}
```

Figure 6 ir_receive.c updated

The following screenshot shows a sample output from the updated ir_receive code. Try to identify the signals present; the screenshot shows 3 complete signals.

```
Sample Collection Complete
Raw Received Value
11000000000000000000000001111111111111111111111000000000000000000000111111111111
11111111111000000000000000000000001111111111111111111111100000000000000000000111
11111111111111111110000000000000000000001111111111111111111100000000000011111111
11111111111111111111111100000000000111111111111111111111111111111111111111111111
00000000000000000000000001111111111111111111111100000000000000000000011111111111
11111111111100000000000000000000001111111111111111111111100000000000000000000011
11111111111111111111100000000000000000000000001111111111111111111111000000000001
11111111111111111111111111111111111111110000000000011111111111111111111111111111
11111111111111111111100000000000000000000000011111111111111111111100000000000000
00000000011111111111111111111111110000000000000000000000001111111111111111110000
00000000000000011111111111111111111111100000000000000000000001111111111111111111
11100000000001111111111111111111111111111111111111111100000000001111111111111111
1111111111111111111111111111100000000000
```

Figure 7 ir_receive.c sample output signal

```
Sample Collection Complete
Raw Received Value
11000000000000000000000001111111111111111111111000000000000000000000111111111111
11111111111000000000000000000000001111111111111111111111100000000000000000000111
11111111111111111110000000000000000000001111111111111111111100000000000011111111
11111111111111111111111100000000000111111111111111111111111111111111111111111111
00000000000000000000000001111111111111111111111100000000000000000000011111111111
11111111111100000000000000000000001111111111111111111111100000000000000000000011
11111111111111111111100000000000000000000000001111111111111111111111000000000001
11111111111111111111111111111111111111110000000000011111111111111111111111111111
11111111111111111111100000000000000000000000011111111111111111111100000000000000
00000000011111111111111111111111110000000000000000000000001111111111111111110000
00000000000000011111111111111111111111100000000000000000000001111111111111111111
11100000000001111111111111111111111111111111111111111100000000001111111111111111
1111111111111111111111111111100000000000
```

Figure 8 ir_receive.c sample output signal 3 messages highlighted

Appendix 1: IR Receiver Reception Distance

One important consideration in an IR communication system is its effective range in actual operating conditions. The effective range can be adjusted by changing the value of the current limiting resistor at the IR emitter. Using the hardware configuration from the IR Receiver Kit tutorial “Hardware Testing” section, it is easy to experimentally test the effective range of the IR system. Figure 1 below shows the testing configuration:

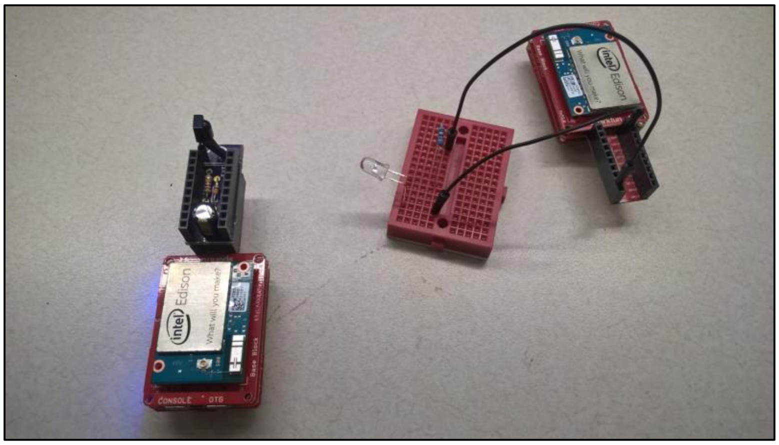

Figure 1 IR range testing setup.

In addition to the items in this setup, a ruler, meter stick, or yard stick should be used to accurately measure the distance between the receiver and the emitter. This graph shows the Max Distance VS Current using the TSOP38238 IR Receiver Diode and the Infrared 950nm LED available from SparkFun.

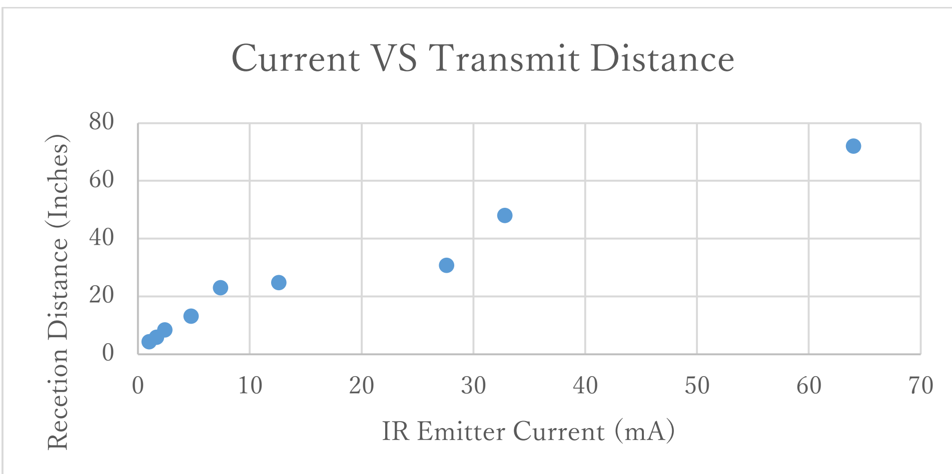

Figure 2 Experimental Range Data

References

1. https://learn.sparkfun.com/tutorials/ir-communication
2. http://www.sbprojects.com/knowledge/ir/