

# HOW TO DECODE IR SIGNALS TO CONTROL A WALKING MACHINE

UCI Walking Critters Project 2020-21 (version 3)

Ronald P. Kessler, Ph.D.

## Table of Contents

INTRODUCTION .....	2
Statement of the Problem .....	2
Understanding IR Streams .....	3
Creating a Binary Stream of Bits from a Decimal Number .....	6
Decoding IR Streams: Example Oscilloscope Patterns.....	7
APPENDIX A. Bit patterns for standard directional control .....	8
APPENDIX B. Sony IR codes for standard remote buttons .....	9
APPENDIX C. Microcontroller Code Examples for a Simple IR Controller.....	10
Parallax Basic Stamp 2 Version .....	10
Basic Stamp 2 IR Wiring Setup for Oscilloscope .....	11
Arduino UNO Version.....	12

## INTRODUCTION

Welcome to the UCI Walkers research project for 2020. Graduate students in the mechanical engineering department have been busy building two and four-legged walking machines. They are affectionately called “Critters”. You can see some in action [here](#). The goal of the project is to introduce students to the theory and application of 4-bar linkages. This mechanical design can be used as a “leg” on a critter to make it walk.

In the animation, the point at the bottom of the triangle piece represents the “foot” and the purple line shows the curving motion the foot takes. This is called a coupler curve. The foot moves in a straight line just like our feet move when we walk. Then, just as we do, the foot is lifted and thrust forward until it contacts the ground again. While the foot is in contact to the ground it propels the critter along its path. The trick is to make it walk in a realistic and stable way. To view some real-life applications of linkages, click [here](#).

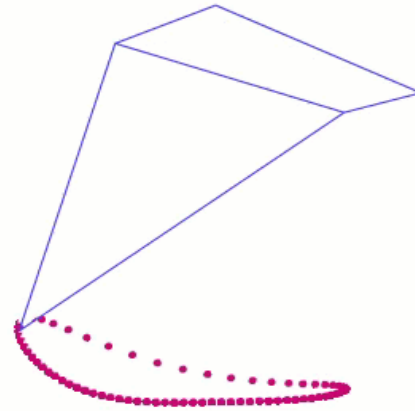


Figure 1: Example of 4-Bar linkages used to make a leg for the walkers.

## Statement of the Problem

As you might imagine, designing and manufacturing a walking critter is a big challenge. There are several versions of linkages to choose from. However, making one of these critters turn is even more difficult. My hypothesis was “can motion and turning used in robotics be adapted to the critter under construction”? To test this idea, a Sony® TV remote control, Infrared Sensor, and Arduino Uno microcontroller were configured to allow the user to move the critter forward, backwards, left, and right. The good news was that it can! The rest of this report will focus on how the IR signals from the TV remote can be decoded and not on the actual design and construction of the critters themselves.

IR streams are not that dissimilar to other wireless technologies such as Bluetooth. To the computer, the data is just an organized package of binary bits. The IR energy spectrum (Figure 1) is outside the visible light spectrum and is at a lower frequency than red light. That is why humans cannot see it.

Now, let’s take a closer look at how the pattern of electrical signals can be decoded.

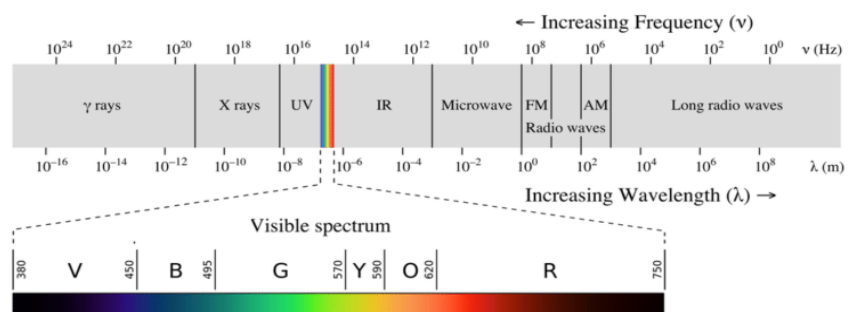


Figure 2: IR Radiation in relation to visible light & electromagnetic spectrum.

## Understanding IR Streams

When the Sony® IR transmitter sends a signal from the remote, a series of 12 bits is streamed to the receiving device. The IR operates on a frequency of 38.5Khz. When nothing is being transmitted, the output of the sensor is high (5 volts). As soon as a signal is sent, the IR sensor voltage drops to zero. The first bit is the widest and represents a “start bit”. You can see in Figure 4 that the width of that pulse is 2.4ms. Then the next seven (7) pulses represent the actual code for the key that was pressed.

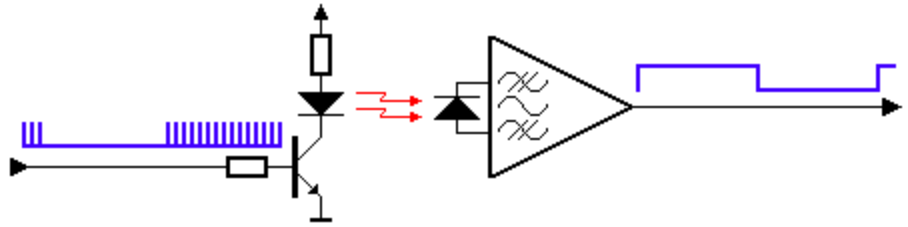


Figure 3: IR Transmitter/Receiver Model

The remaining five pulses contain a binary value that specifies whether the message is intended for a TV, VCR, CD, or DVD player. It is important to note that the least significant bit (LSB) is sent first (Little Endian).

The Sony IR protocol is based on a pulse-width signal coding scheme. The width of the pulse determines its binary value. A .6ms or (600µs) pulse equates to a logical 0 and a 1.2ms (1200µs) pulse defines a logical 1. Not all companies use the same protocol, so it is important to know the coding scheme for the TV remote you happen to be using.

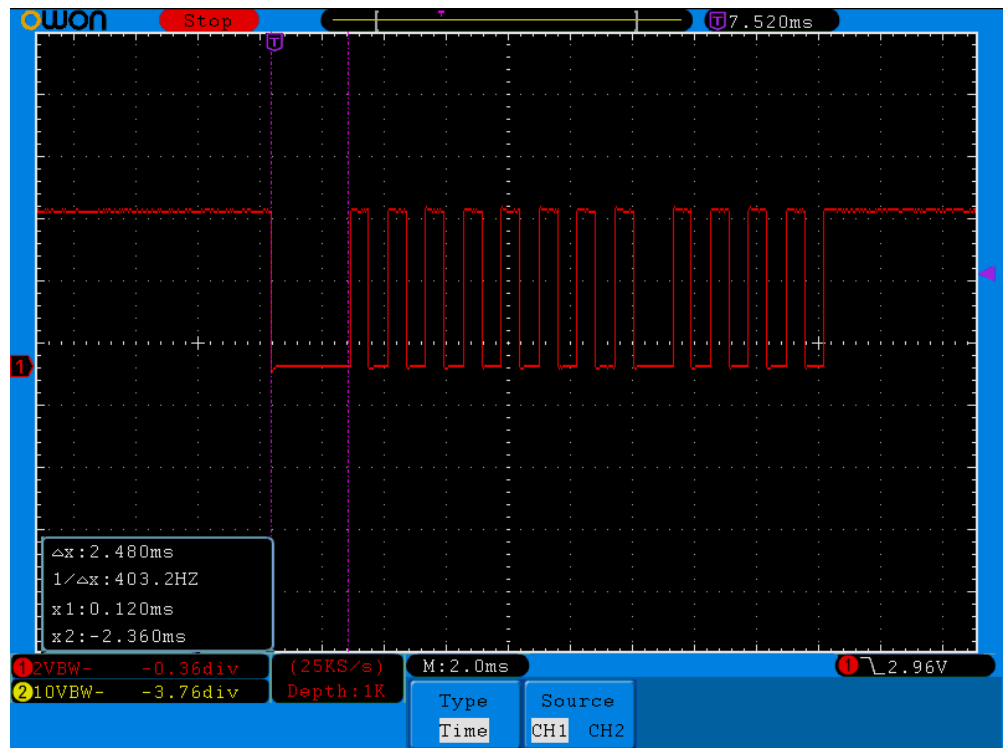
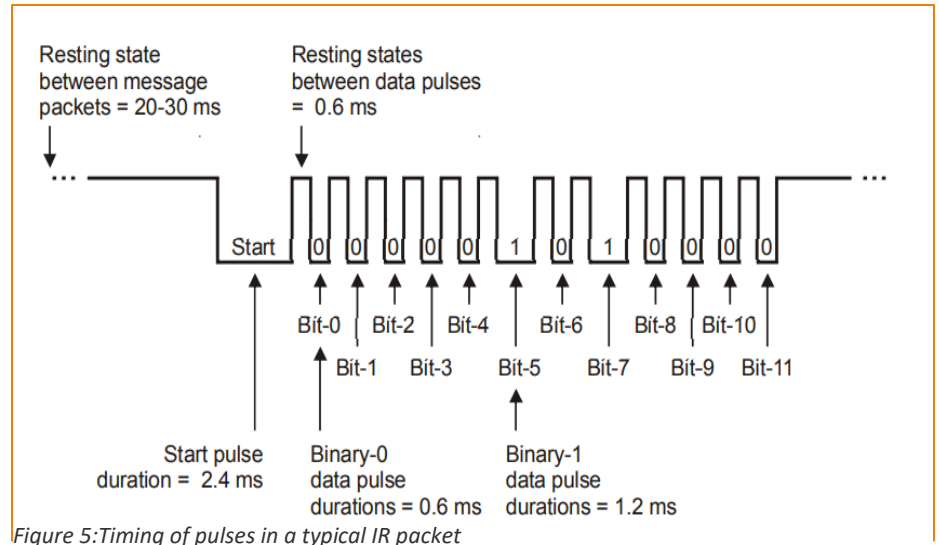


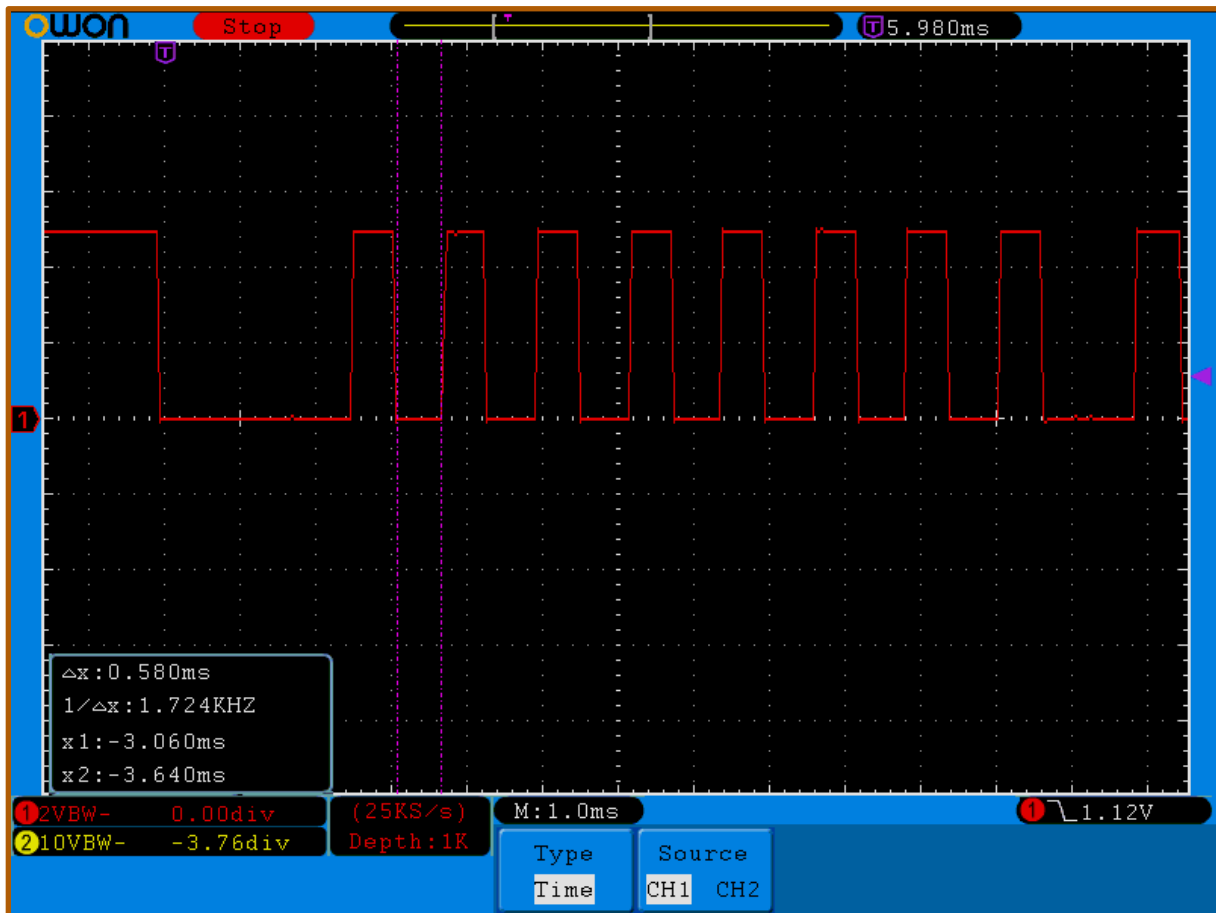
Figure 4: IR 12-bit Packet. The wide pulse is ~2.5ms and is called the Start Bit.

Figure 5 summarizes the format of a typical packet. We will only be concerned with bits 0-6 because those define the direction keys on the TV remote.

Image from *IR Remote for the Boe-Bot version 1.1* by Andy Lindsay, Parallax



In figure 6, we can see the width of a low (logical 0) bit is approximately .6ms wide.



In Figure 7, we can see the width of a logical 1 bit is approximately 1.2ms wide.

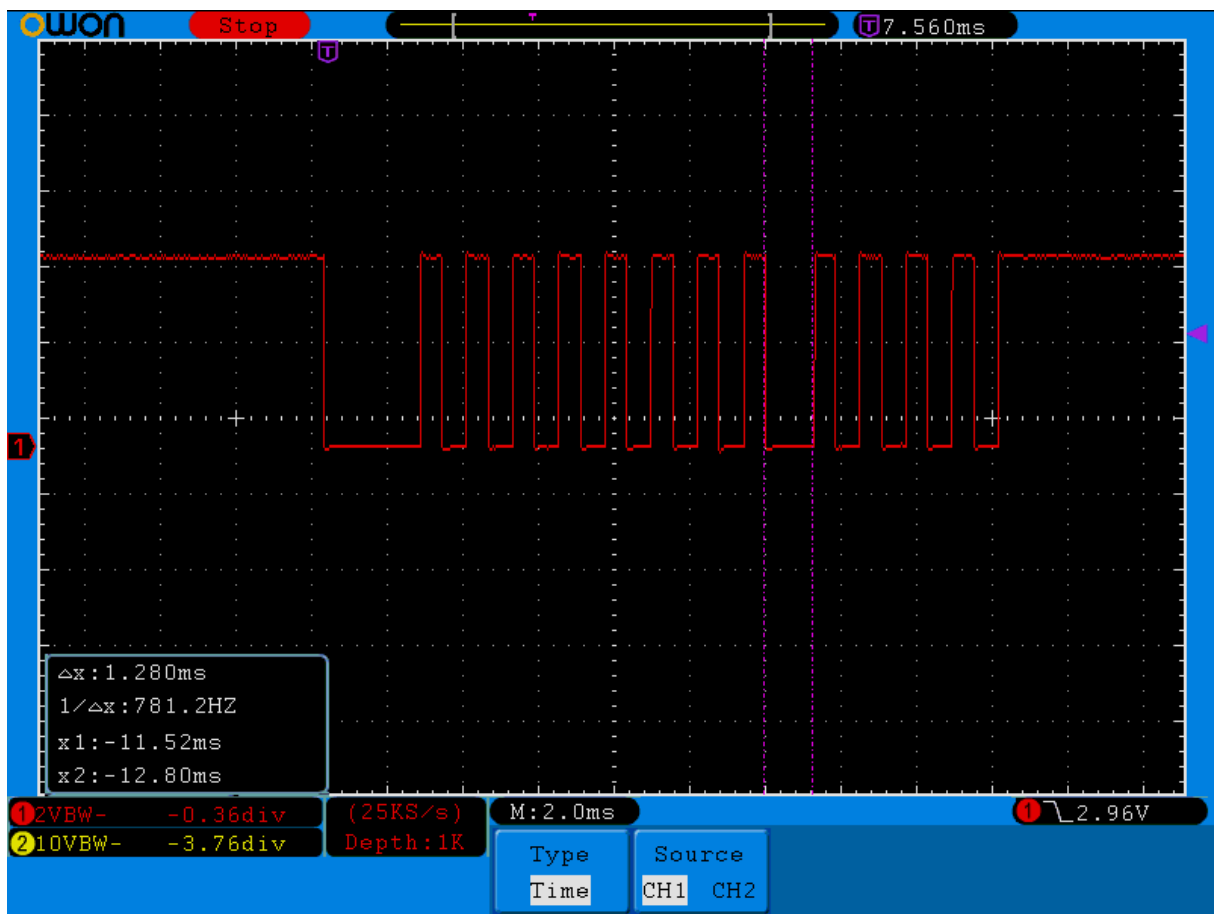


Figure 7: Logic High bit is 1.2ms wide

## Creating a Binary Stream of Bits from a Decimal Number

Before we can decode any data stream between devices, we must make sure we understand the binary numbering system. Since electricity has only two states (on/off), computers and other devices were designed using this numbering system.

Look at figure 8. Before we can transmit/receive IR data, we must package up our values to be in binary format. Assume the Channel UP button on my remote sends the code 53 in decimal. The number 53 = 00110101 in binary. Binary works just like turning a light switch on or off in your home. There are only two choices: on or off. There is not any in between. Look at the purple row below. Each box represents bit (binary digit).

There are eight (8) bits which makes up one (1) byte. The first purple box on the right is the called the least significant bit (LSB). Think of this as the 1's column. The first purple box on the left is the most significant bit (MSB).

Binary	0	0	1	1	0	1	0	1	
Multiplier	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$	
Decimal	128	64	32	16	8	4	2	1	
			32	16		4	1		
			+ 16 + 4 + 1 =					53	

Figure 8: Binary to Decimal Conversion

The decimal row shows the value of each column. As you can see, there is a "1" in the 32 column, a "1" in the 16's column, a "1" in the 4's column and a "1" in the 1's column. If we add these values up, ( $32+16+4+1 = 53$ ), we get the decimal equivalent of a binary number.

When we press a button on the remote, each button has a code from the factory just like your keyboard does. When you type "A", it is converted to the number 65. The "B" key = 66, and so on. So, if I press **Channel Up** on a Sony remote, the number 16 is transmitted to the IR receiver. However, the number 16 must first be converted into its binary equivalent and each individual bit must be transmitted in exactly the correct order. If we use Figure 8 as a reference, the number 53 must be transmitted as 00110101. You can clearly see this in Figure 7. The string of pulses is called a *stream*.

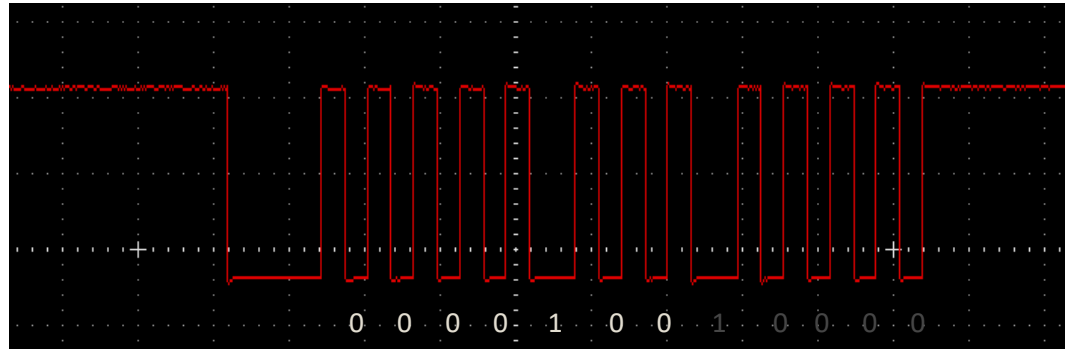
Next, we will look at the oscilloscope images and see if can decode the stream. Keep in mind, we are looking at the width of the pulse that is a logical 0. This means, the light switch is off for about .6ms. When you observe a logical 1, the light switch is on and it remains on for 1.2milliseconds. A millisecond is 1/1000 of a second. When no data is being transmitted, the signal stays "high" (5 volts in our case). You will also notice .6ms high pulses between the bit patterns. This allows the timing between pulses to be precise. Without this precision, the microcontroller cannot decode the message!

One last point, the Arduino and Basic Stamps microcontrollers transmit the LSB first. This is called "Little Endian". The rules we are learning about are called protocols. They are agreed upon standards that companies can use to standardize communication between devices. This applies to Bluetooth, wireless, ethernet, and cellular standards.

## Decoding IR Streams: Example Oscilloscope Patterns

Let's start with the **Channel Up** button.

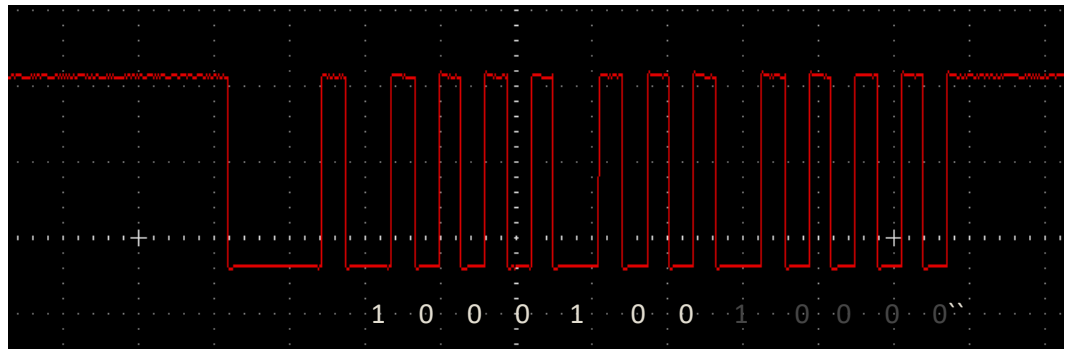
When it is pressed, it acts as the **FORWARD** button and it transmits this pattern. Let's try and decode it.



Remember the LSB is sent first so in binary this is 0 0 1 0 0 0 = 16 in decimal. When the scope displays a stream, the first bit is on the left and each bit after that shows up farther to the right. Think of the horizontal scale as time. So, the first 7 bits (0000100) must be re-arranged so we can determine their binary values (0010000).

Let's try one more so you can get the hang of it. This pattern represents the **BACK** key or **Channel Down**.

The first 7 bits are sent in the order of 1 0 0 1 0 0. But we need to reverse their order. So, this time the code is 0 0 1 0 0 1 = 17 in decimal.

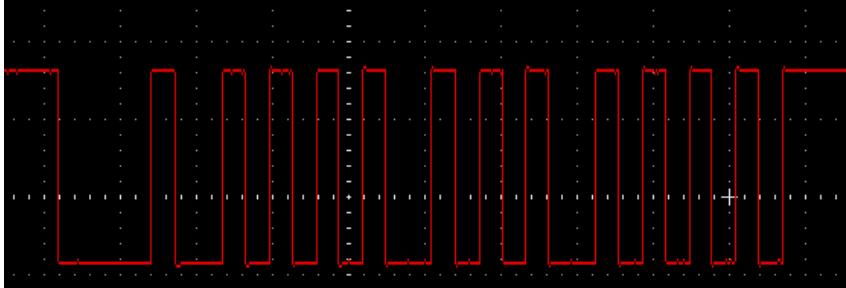
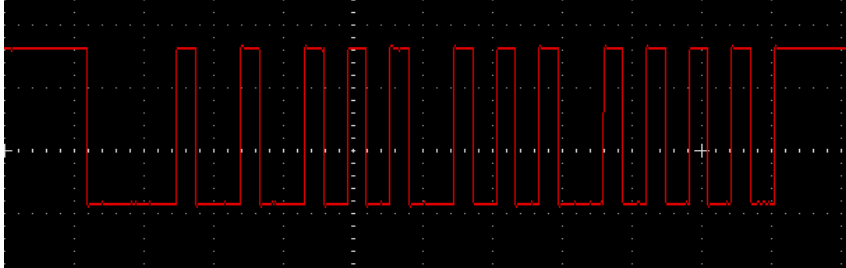


The appendices that follow will serve as a reference as you learn to program your devices. Remember, no two manufacturers use the same numbers for their keys, they use different protocols, and they do not always transmit bits using little endian. Some companies use "Big Endian" where the MSB is transmitted first.



## APPENDIX A. Bit patterns for standard directional control

Scope images of the bit patterns for Channel Up/Down and Volume Up/Down. Remember, the bit on the left is the LSB (Bit 0) so you must reverse the pattern to decode it. Ignore last 5 bits and read right-to-left as you write down the bit values. Also, the bits that are low are the ones we decode. When the signal is high (5VDC) no IR signal is being transmitted.

<p>0 0 0 0 1 0 0 = 0010 000</p> 	<p>Channel UP Button</p> <p>0010 000 = 16 Decimal</p>
<p>1 0 0 0 1 0 0 = 0010 001</p> 	<p>Channel DOWN Button</p> <p>0010 001 = 17 Decimal</p>
<p>0 1 0 0 1 0 0 = 0010 010</p> 	<p>Volume UP Button</p> <p>0010 010 = 18 Decimal</p>
<p>1 1 0 0 1 0 0 = 0010 011</p> 	<p>Volume DOWN Button</p> <p>0010 011 = 19 Decimal</p>

## APPENDIX B. Sony IR codes for standard remote buttons

**SONY IR REMOTE CODES for UCI Critters Project**

Command Key	BINARY STREAM	Decimal Code
<b>1</b>	0000 000	0
<b>2</b>	0000 001	1
<b>3</b>	0000 010	2
<b>4</b>	0000 011	3
<b>5</b>	0000 100	4
<b>6</b>	0000 101	5
<b>7</b>	0000 110	6
<b>8</b>	0000 111	7
<b>9</b>	0001 000	8
<b>0</b>	0001 001	9
<b>Enter</b>	0001 011	11

**SONY IR REMOTE CODES for UCI Critters Project**

Channel UP	0010 0 <b>00</b>	16
Channel DOWN	0010 0 <b>01</b>	17
Volume UP	0010 0 <b>10</b>	18
Volume DOWN	0010 0 <b>11</b>	19

**NOTES:**

Bits are transmitted Little Endian

Logical 0 = .6ms (600us) on and .6ms off

Logical 1 = .12ms (1200us) on and .6ms off

All bits are separated by .6ms interval

We only need to test bits 0 & 1 to determine which button was pressed. Look at the red bits in the gray box above. That represents the four combinations of 2 bits taken 2 at a time. Can you see why the 1-4 keys respond as well?

## APPENDIX C. Microcontroller Code Examples for a Simple IR Controller

### Parallax Basic Stamp 2 Version

' Control your Boe-Bot with an IR remote set to control a SONY TV  
' with the 1-4 or CH+/- and VOL+/- keys.

```
{ $STAMP BS2 }
{ $PBASIC 2.5 }

time    VAR    Word(2)    ' SONY TV remote variables.
CW      CON    1300
CCW     CON    1700
HALT    CON    1500

DO
DO      ' Beginning of main loop.
        ' Wait for rest between messages.
    PULSIN 9, 1, time(0)
    LOOP UNTIL time(0) > 2000

    PULSIN 9, 0, time(0)    ' Measure/store data pulses.
    PULSIN 9, 0, time(1)

    ' Decide which maneuver to execute depending on the combination
    ' of pulse durations stored in the first two pulse measurements.

    IF (time(1) < 1000) AND (time(0) < 1000) THEN
        GOSUB GoForward
    ELSEIF (time(1) < 1000) AND (time(0) > 1000) THEN
        GOSUB GoBack
    ELSEIF (time(1) > 1000) AND (time(0) < 1000) THEN
        GOSUB TurnRight
    ELSEIF (time(1) > 1000) AND (time(0) > 1000) THEN
        GOSUB TurnLeft
    ELSE
        GOSUB DoStop
    ENDIF
    LOOP

GoForward:
    PULSOUT 13, CCW    ' Forward
    PULSOUT 12, CW
    RETURN

GoBack:
    PULSOUT 13, CW    ' Backward
    PULSOUT 12, CCW
    RETURN

TurnRight:
    PULSOUT 13, CCW    ' Right rotate
    PULSOUT 12, HALT
    RETURN

TurnLeft:
    PULSOUT 13, HALT    ' Left rotate
    PULSOUT 12, CW
    RETURN

DoStop:
    PULSOUT 13, HALT
    PULSOUT 12, HALT
    RETURN

END OF BS2 CODE
```

## Basic Stamp 2 IR Wiring Setup for Oscilloscope

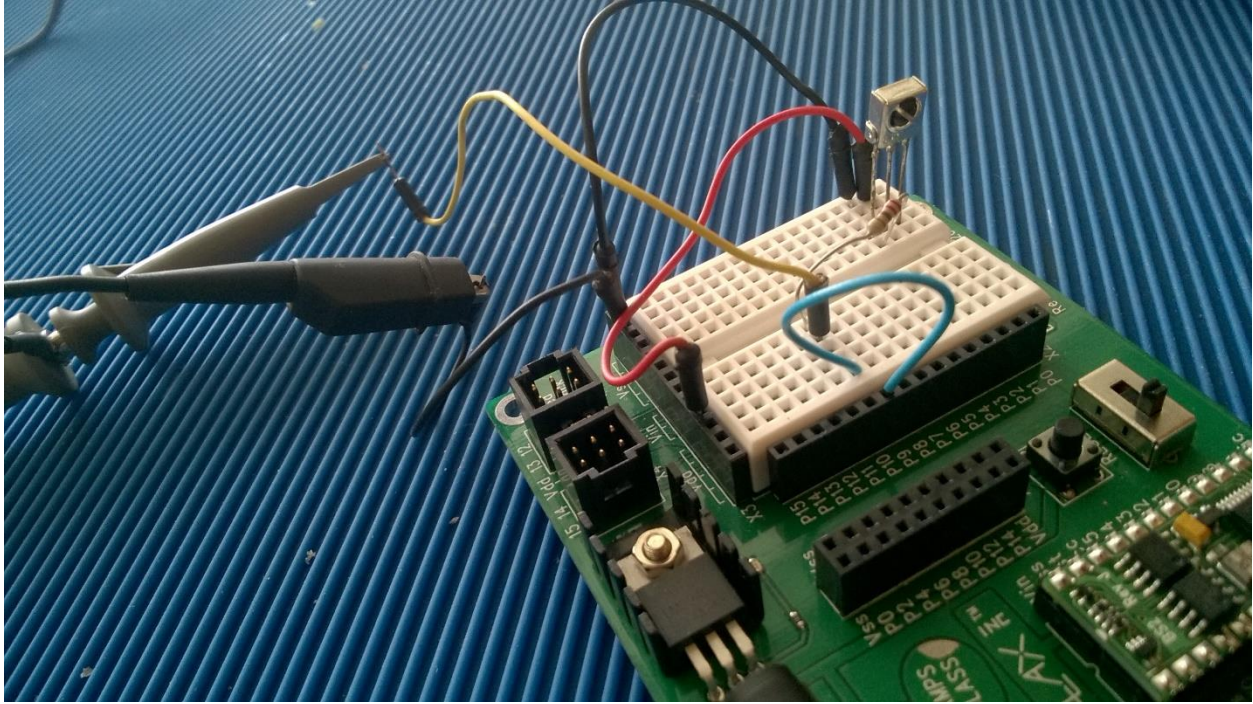


Figure 9: Setup for Displaying IR Streams on Scope

- Scope Settings: Channel A
  - DC Coupling
  - Vertical 2v/div
  - Horizontal 2ms/div
  - Trigger Mode: Pulse
  - Auto Trigger -80mV trigger level
  - Probe Pin 9 (after resistor)
- IR Sensor Bulge Side
  - Left Signal 220 $\Omega$  to pin 9
  - Center VSS (Ground)
  - Right VDD

## Arduino UNO Version

```

/* PURPOSE:
* DEMO HOW TO DETECT SONY REMOTE IR KEY CODES
* See my comments in loop function below. You can use either decode method.
* Entire 12 bits    First byte codes
* Channel Up = 144  16
* Channel Dn = 145  17
* Volume Up = 146   18
* Volume Dn = 147   19

```

Using the IR receiver in your study group kit:

1. Place the IR module in a breadboard with the sensor facing you.
2. Connect a green wire from the leftmost pin to GND on Arduino.
3. Connect a RED wire from the center pin of the sensor to 5V on Arduino.
4. Connect another wire from the right pin to pin 2 of the Arduino. This is the signal pin.
5. Make sure you have the IRRemote library installed on your computer. I use ver 3.3 as of this writing.
6. Save & upload your code
7. Open the serial monitor window. TOOLS | SERIAL MONITOR. Make sure to set baud rate to 9600.
8. Point your Sony remote towards the sensor and push the buttons. The button code should appear in the monitor window.
9. WRITE YOUR CODES DOWN AND INCLUDE THEM IN YOUR SKETCHES!!!

```

*/
//***** START CODE HERE *****

//---1. now add the library
#include <IRremote.h> //ver 3.3 5/14/2021

//---2. define our sensor data pin and read first byte
const byte IR_RECEIVE_PIN = 11;

void setup()
{
  //---3. open the serial port and monitor at 9600 baud rate.
  Serial.begin(9600);

  //---4. Start our IR receiver on the arduino. We pass in two arguments: pin# and system command
  // to ignore blinking the onboard LED when a message is received.
  IrReceiver.begin(IR_RECEIVE_PIN, DISABLE_LED_FEEDBACK);
}

void loop()
{
  //---if data is beig received then decode it, print to the monitor, and keeping listening for more!
  if (IrReceiver.decode())
  {
    //---use command to use first byte as described in my report
    Serial.println(IrReceiver.decodedIRData.command); //only the first byte!

    //---use this line to decode entire 12bits
    //Serial.println(IrReceiver.decodedIRData.decodedRawData); //to see entire stream
    IrReceiver.resume(); // Receive the next value
  }
}

```