Lab 13: Microcontrollers II

You will turn in this lab report at the end of lab. Be sure to bring a printed cover-page to attach to your report.

Prelab

- Watch this video on DACs
  https://www.youtube.com/watch?v=b-vUg7h0lpE.
- Read this article on the circuit analysis of a R-2R DAC.
  https://www.tek.com/blog/tutorial-digital-analog-conversion-r-2r-dac
- Q. For a 3-bit R-2R DAC with inputs of zero to five volts, how many output voltage levels can be created and what is the maximum output voltage?
- Q. For a 5-bit R-2R DAC with inputs of zero to five volts, how many output voltage levels can be created and what is the maximum output voltage?
- Install the IRremote library via the following steps:
  - In the Arduino IDE go to `Sketch -> Include Library -> Manage Libraries...`
  - Search for "IRremote" (the author should be "shirriff")
  - Click on the library and an Install button should appear. Install the latest version if it's not already installed.
  - You should now see around a dozen examples by going to `File -> Examples -> IRremote`
- From Moodle, download and unzip the file `lab13code.zip` to anywhere you normally keep files. Inside should be two folders, `piano` and `simple_dac`. Nothing else needs done with these till lab.

Supplies

- SparkFun Inventors Kit (SIK)
- SparkFun IR Control Kit
- Multimeter
- Oscilloscope
- Resistors: 1kΩ(x14), 10kΩ(x2), 330Ω(x1)
- Capacitors: 0.1uF(x1) and 1uF(x1)
- 2N3904 or 2N2222 NPN transistor
- Photoresistor (GL5516)
- Speaker

Part I: IR communication

1.1 Theory

One common use of microcontrollers is in systems which have IR remote controls such as DVD players, stereo systems, TVs and the like. Both in the remote and in the receiving piece of equipment there will be a microcontroller to either generate and send, or receive and translate the logic codes which are transmitted via pulses of infrared light. We're going to use the arduino in this fashion, but first let's learn a bit about how IR communication works.

Your IR detector is a bit more than just a simple photocell. It includes a tiny microchip which converts incoming IR signals to digital outputs. You might think that a signal would be just blinking the LED on or off, much like Morse code, however it is somewhat more complicated than that. When the LED is transmitting it is actually sending out a PWM signal at 38 kHz[1]. The microchip in your IR receiver demodulates this signal into digital highs and lows which can be read by the Arduino.

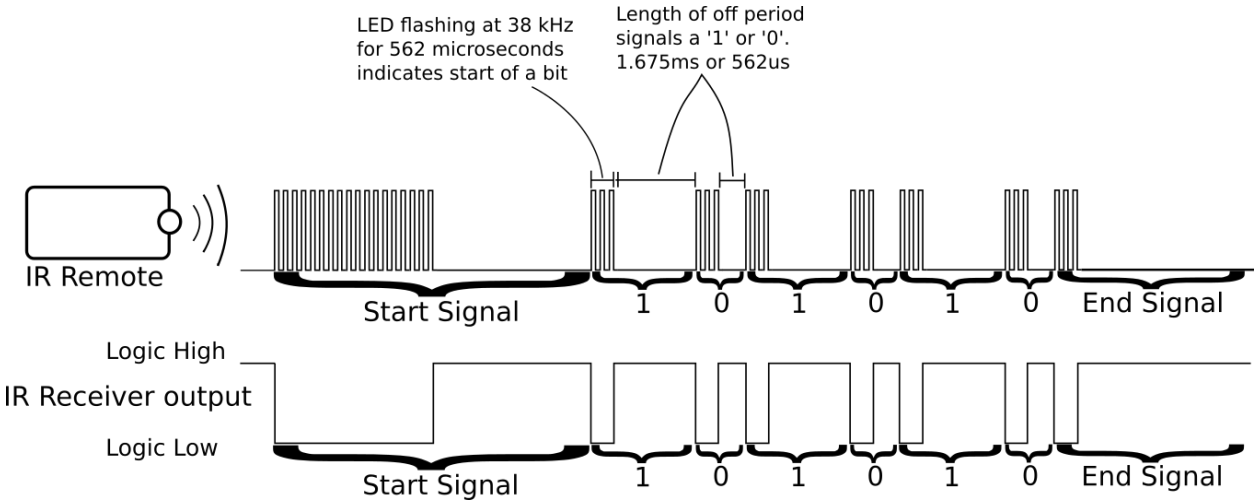


Figure 1: The relationship between the output of the IR LED and the output of the IR receiver. **Counter-intuitively, a logic LOW corresponds to the time during with the remote IS transmitting.** *Note that the spacing is not quite to scale.*

There are various protocols for IR signals, the one your remote uses is the NEC protocol. See figure 1 for an example of how the NEC protocol works. Each 'one' or 'zero' begins with a $562\mu s$ logic low pulse. To signal a 'one' this is followed by a $1.675ms$ logic high while to signal a 'zero' it is followed by a $562\mu s$ logic high.

Q. How much longer does it take to send a 'one' than a 'zero'?

The full data packet consists of a start signal, 32 bits of address and data, and a stop signal. See figure 2 for an example data packet. You can see that there are 32 'ones' and 'zeros' which can then be converted from a 32 bit binary number to a hexadecimal value. Each button on your remote sends out a unique hex value.

[1]This is one of the most common frequencies but there are others, any given IR receiver is tuned to be most sensitive to a specific frequency.

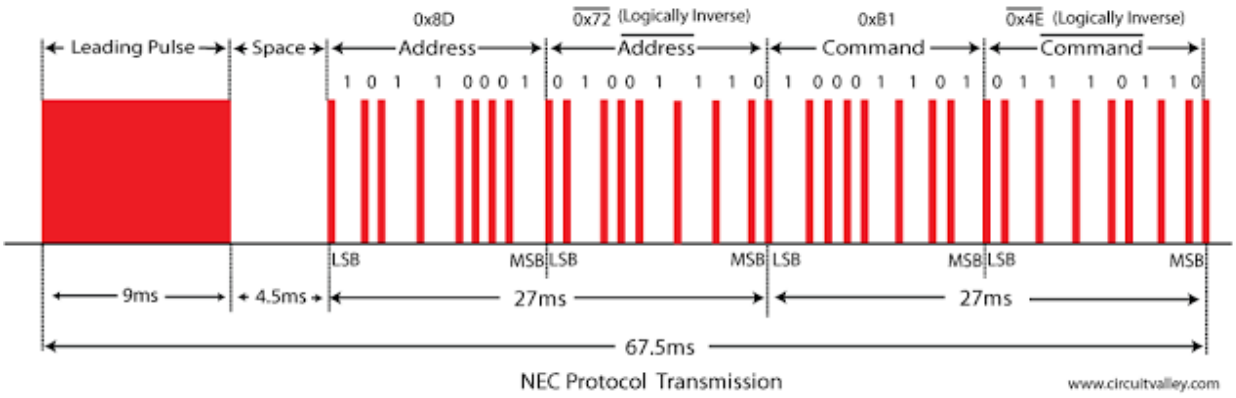


Figure 2: **Note that the red is when the LED is transmitting and is actually when the receiver is outputting a logic LOW, as you will see on your scope.**
Source: www.circuitvalley.com/2013/09/nec-protocol-ir-infrared-remote-control.html

1.2 Manual Decoding

You are going to use the oscilloscope to capture the output of the IR receiver and check that the remote is sending out the expected codes. The expected values can be seen in figure 4

- Using your bench PSU, connect pin 3 of your IR sensor to 5 volts, pin 2 to ground/low, and pin 1 to your scope probe. (*See figure 3 for the pin out.*)

When you connect your scope you should see the a constant five volt signal.

- Zoom in to 10ms/div on the time axis and set your trigger between zero and five volts.

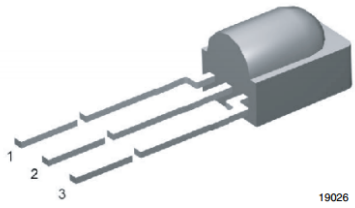
When you push a button on your remote you should see a signal similar to that in figure 2 show up on your screen.

- Scroll the time axis until the whole signal appears when you trigger the remote.
- Now use the "SINGLE SEQ" button to capture and hold one data packet.

At this zoom level it is hard to differentiate between ones and zeros.

- Zoom in on the time axis of the captured packet and translate the signal into a 32 bit binary number.
- Convert this number to hexadecimal.
  (I suggest using wolframalpha.com. For example, typing in: "10101010 bin to hex" should result in a hex value of aa_{16}.)
- Check that your result matches the button you pressed by comparing it to figure 4

Q. Record the binary and hexadecimal number in your report along with which button you pressed.



MECHANICAL DATA
Pinning for TSOP382.., TSOP384..:
1 = OUT, 2 = GND, 3 = V_S

Figure 3: Datasheet for the IR receiver www.vishay.com/docs/82491/tsop382.pdf

1.3 Arduino Decoding

Now we're going to make things easy and decode the other buttons using Ken Schriff's IR library which you installed in the prelab.

- Connect pin 1 of your IR Sensor to pin 11 on the Arduino.
- Connect pins 2 and 3 to the Arduino ground and 5 V rails.
- Open `File` → `Examples` → `IRremote` → `IRrecvDemo` and upload the code.
- Open the serial monitor. Point the remote at the sensor and push each button in turn. Check the hex value in the serial output against figure 4.

Q. Do all the values agree with the labels in figure 4?

Q. If you hold down one button what do you see? Google the NEC IR protocol and explain what is happening.

Figure 4: Remote codes

Part II: Waveform Synthesis

2.1 Background Info

The Arduino board you are using has the Atmega 328p microcontroller which is only capable of outputting zero or 5 volts on its output pins. Even the so called `analogOut()` function really just creates a square wave with a variable duty cycle which can mimic some of the effects of varying the voltage. To be able to actually create an output voltage between zero and five volts requires some extra circuitry known as a Digital to Analog Converter or, more commonly, a DAC.

A very simple DAC is the R-2R ladder. The total number of output voltage levels (including zero volts) is 2^n where n is the number of stages (bits) in the "ladder." The output voltage is give by:

$$V_{max} = (\text{Logic High Voltage}) \times \sum_{n=0}^{\text{max bit}} \frac{b(\text{max bit} - n)}{2^{n+1}} \tag{2.1}$$

where b(x) is a function which returns either zero or one depending on if the xth bit is high or low. For example, the max output voltage of a 4-bit R-2R DAC with logic levels of 5 volts is:

$$V_{max} = 5 * \left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16}\right) = 4.6875 \text{ V} \tag{2.2}$$

The output voltage can be represented as a binary number where the most significant bit (MSB) contributes the greatest amount to the output voltage. For example, the binary number 1100 (12 in base 10) would give the following voltage:

$$V = 5 * \left(\frac{1}{2} + \frac{1}{4} + \frac{0}{8} + \frac{0}{16}\right) = 3.75 \text{ V} \tag{2.3}$$

2.2 Building a 3-bit DAC

To test your understanding of the DAC you will first construct a 3-bit DAC and manually connect the three inputs (bits) to ground or five volts.

- Using only 1% 1kΩ resistors construct the circuit shown in figure 5 (use two 1 kΩ resistors in parallel to get 500Ω).
- Connect all three bits (B0, B1, B2) to five volts. (Use your bench power supply set to 5 volts as measured with your multimeter.)

Q. Using a multimeter, measure the output voltage and compare to what you calculated in the prelab.

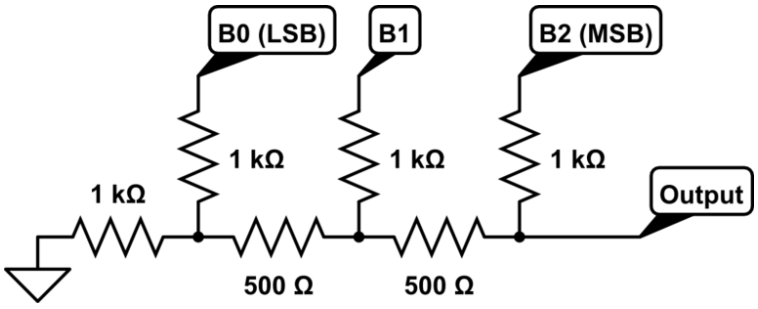


Figure 5

Notice that the least significant bit (LSB) is located on the left side of the diagram, while when you write out a binary number the LSB is the rightmost digit.

Q. Calculate what output voltage you expect for a binary input value of 110_2 (6 in base 10).

- Connect the bits to zero and 5 volts as appropriate (again, realize that the MSB bit has a value of 1 and LSB has a value of 0.)

Q. Measure the output voltage.

Q. What binary value corresponds to the voltage one step below 110_2? Calculate and measure that voltage.

Q. What is the "step size" (i.e. the minimum jump in voltage), both theoretical and measured?

This DAC is very course grained, in the next sections you will add a 2 more bits which will significantly increase the resolution.

2.3 Using a DAC with a microcontroller

The first program you will run simply steps the DAC output voltage from zero up to the max output. Figure 6 shows the output for a 3 bit DAC. Take a minute to understand what is going on in the figure.

- The topmost trace corresponds to bit zero (B0) in the 3 bit DAC you built. You can see that this least significant bit alternates values at each step.
- The most significant bit (B2) is low for the first half of the ramp and high for the second half. In general, the n^{th} bit alternates every 2^n steps.
- Look at each column of numbers. At each step the states of the top three traces form the binary value indicated below each step of the bottom trace. The topmost trace corresponds to the rightmost digit. The base 10 (decimal) representation of that base 2 (binary) value is shown above each step.

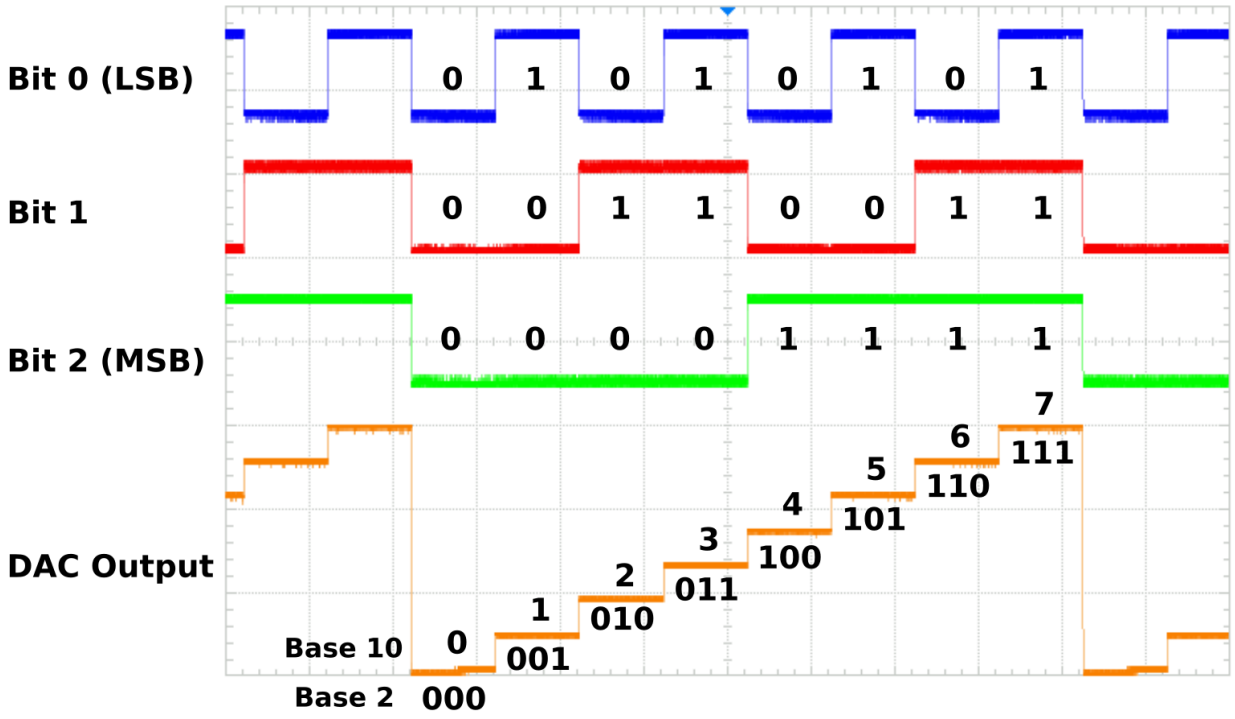


Figure 6: Each input and the output from a 3-bit R-2R DAC

2.4 Building a 5-bit DAC

- Construct the circuit shown in figure 7 and connect to the indicated pins on your microcontroller board. **You will be using the circuit for the rest of lab.**

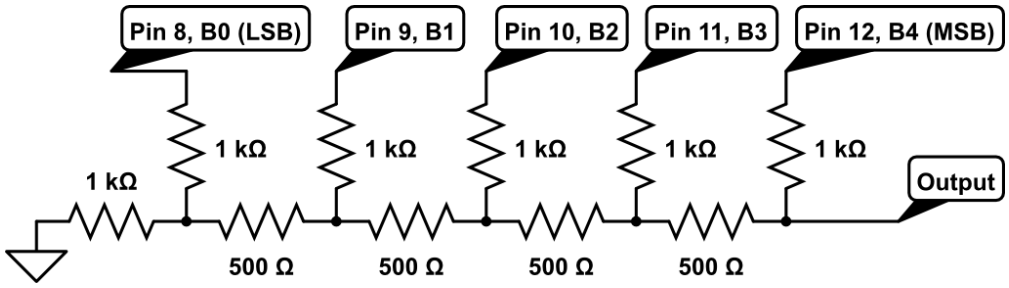


Figure 7: A 5-bit R-2R Ladder DAC

- Download `lab13code.zip` from Moodle and extract the contents. Open the sketch `simple_dac.ino`.

Before you upload the code, read through the following explanations and the comments in the code itself.

```
#define maxDACValue 32
```

The `#define` macro replaces all occurrences of `maxDACValue` with `12`. This is different from a variable since this happens before the code is even compiled.

```
void setup(){
  DDRB = DDRB | B00011111;
  PORTB = PORTB & ~(B00011111);

  Serial.begin(9600);
}
```

Previously, the function pinMode() was used to set a pin as an input or output. Inside the pinMode() function the Data Direction Register (DDR) is being written to, just as is being done explicitly here. Registers are special areas of memory which store various settings for the microcontroller. Each register is a single byte (8 bits) and so can hold settings or data for up to 8 pins. The digital and analog pins on the microcontroller are more generally known as "General Purpose Input/Output" (GPIO) pins. Groups of pins which share a set of registers are known as a "port." Digital pins 8-13 are labeled "port B." These pins are also known as PB0 - PB5. Pin 8 is the same as pin PB0 which is associated with bit 0 (the rightmost bit) in each port B register.

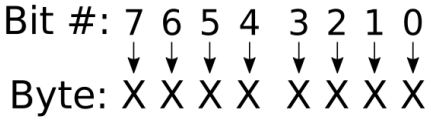
To confuse things a bit, the PORTB keyword in the code above actually refers to the "data register" associated with the collection of pins known as port B. Each bit in the data register controls whether a pin is high (5 V) or low (0 V). Directly writing to this data register is known as "port manipulation" and is sometimes required to achieve the desired performance. This method is not only faster than using the arduino functions digitalWrite() and digitalRead(), but also allows us to set the state of many pins at exactly the same moment. This is very important for setting DAC values.

Setting a single bit to 1 without affecting any other bits.

This is accomplished by using the bit-wise logical OR operation represented by "|" (the "pipe" symbol), found just above the `Enter` key on your keyboard. Don't confuse this with capital I (eye) or lowercase l (ell) or 1 (one).

The output of "OR-ing" two bits is one, unless both bits are zero. Below you can see the result of OR-ing the initial byte with a "bit-mask." This results in bit 4 (the fifth bit from the right) being set to one while all the other bits remain unaffected. Look at each column and check for yourself that the bottom value is the result of the original byte OR-ed with the mask.

```
Byte:  0 1 1 0 1 0 0 1
Mask:  0 0 0 1 0 0 0 0
OR-ed: 0 1 1 1 1 0 0 1 (Byte | Mask)
```



The two ways you'll see this written in code are:

```
byte = byte | mask;
byte |= mask;
```

The second line is just a shortcut way of writing the first line.

Setting a single bit to 0 without affecting any other bits

This is accomplished by using the bit-wise AND operation and the bit-wise inversion of the mask. A single ampersand symbol (&) represents bit-wise AND. Bit-wise inversion (logical NOT) is represented with a tilde (∼). The original byte is AND'ed with the inverse of the mask. The following example shows how to set bit 3 (the forth bit from the right) to zero. Check that the result is the bit-wise AND of the original byte and the inverted mask.

```
  Byte: 0 1 1 0 1 0 0 1
  Mask: 0 0 0 0 1 0 0 0
 ~Mask: 1 1 1 1 0 1 1 1
And-ed: 0 1 1 0 0 0 0 1 (Byte & ~Mask)
```

In code you would see either of the following:

```
byte = byte & ∼mask;
byte &= ∼mask;
```

Again, the two ways of writing it have identical meanings.

Q. Show how to set bit 2 (third bit from the right) of 10111011 to 1.

Q. Show how to set bit 7 (far left bit) of 10111011 to 0. (The more common way to say this would be "Show how to clear bit 7.")

- Upload the simple_dac code and check with your scope that the output looks like a staircase
- Add in the code to make a triangle wave (staircase both up and down).

Q. Write down the code you used in your report.

2.5 Remote Controlled Piano

You will now add the IR controller and use the remote control to select the frequency of a sine wave for the Arduino to generate.

2.5.1 Sine Wave Synthesis

As the video you watched for prelab showed, a DAC can be used to generate a sine wave by stepping through an array of amplitude values. The amplitude values could be calculated on the fly, but the calculations would take up a large part of the microcontroller's processing power. The more common approach is to store a large array of amplitude values (covering one complete wavelength) known as a "look-up table." There are websites which will generate these look-up tables, such as http://www.daycounter.com/Calculators/Sine-Generator-Calculator.phtml.

The look-up table has two parameters. One is the max amplitude which depends on the number of bits in the DAC. Since you are using a 5-bit DAC, the max amplitude should be $2^5 - 1 = 31$. Including zero, this means there are 32 possible states. The other parameter is the number of entries in the table. These correspond to the number of points at which the sine wave is sampled. The key idea here is that to generate different frequencies, different numbers of entries will be skipped.

If the output is updated 100 times per second and the sine table is 100 entries long, then if the output shifts by one entry each time it's updated the frequency would be 1 Hz. To double the frequency one would simply skip every other entry. More generally, the output frequency is given by:

$$f = \frac{\text{Update rate}}{\#\text{ of entries}} * (\#\text{ of elements skipped}) \tag{2.4}$$

The update rate will be set using a timer which will trigger an interrupt. The Atmega 328p microcontroller has 3 built in timer modules which allow can be configured to perform a specific action at set intervals. This action is called an "interrupt" because it interrupts the processor no matter what else the processor is doing at the moment.

- In the lab13code folder you should see a folder called piano, and in that folder the files piano.ino and sine_table.h. Open the piano.ino file in the Arduino IDE. You should see a tab at the top for sine_table.h, take a look at it.

Notice at the top the PROGMEM keyword, this tells the compiler to store this data in flash memory instead of RAM (where an array would be stored by default). This is important since this table is too large to fit in the microcontroller's 2 kB of RAM.

The sine table you will be using has 4000 entries and the microcontroller will be configured to update the output 32,000 times per second. So from above, the output frequency will be:

$$f = \frac{32000}{4000} * \text{skips} = 8 * \text{skips} \tag{2.5}$$

This means that changing the number of skips by 1 changes the frequency by 8 Hz. Clearly that means there are many frequencies you can't hit exactly.

2.5.2 Remote Control

- Connect the IR receiver following the pinout in figure 3, with the output pin on the receiver connected to pin 2 on the Arduino board.
- Upload the piano.ino code and use your scope to check that when you press various buttons on the remote the DAC output looks like a jagged sine wave.

2.5.3 Amplification

You can't drive a speaker directly from the DAC because it has a very high output impedance. In fact for an R-2R DAC the output impedance equals R.

Q. What is the output impedance of your DAC?

The single transistor amplifier you built in a previous lab can come to the rescue.

- Construct the circuit shown in figure 8 and connect the output from the DAC as shown.
- Before connecting a speaker, use your scope to look at the base voltage and the emitter voltage while triggering a sine wave. They should look practically identical.
- Check for the positive and negative markings on the speaker then connect the positive side to the output (emitter) and the negative side to ground.
- To make the sine waves cleaner, connect a 1 μF capacitor between the output of the DAC and ground. This forms a low pass filter.

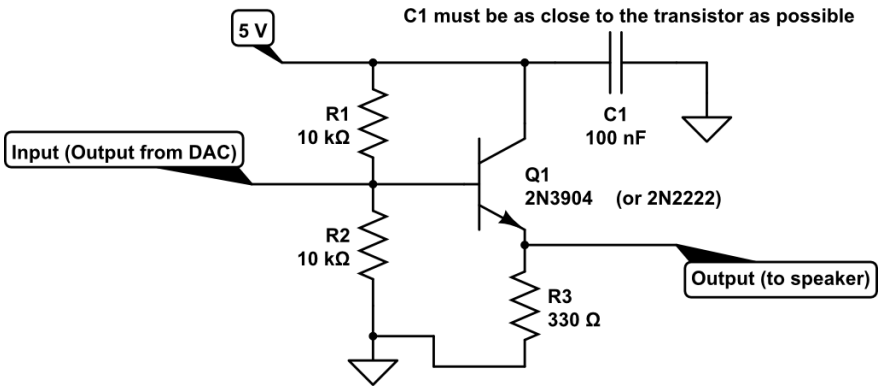


Figure 8: An emitter follower (current amplifier)

2.5.4 Playing Music

- Get a paper cover for your remote control and tape it to the remote (don't use too much tape, you're going to have to remove it at the end.)
- Find where the notes are defined in the code and determine which remote button corresponds to which note (look for the `switch()` statement) and label the paper accordingly.
- Try to play Twinkle-Twinkle Little Star. Show your musical prowess to your instructor! (Note: The small speakers don't handle the fourth octave notes very well which is why the song has been transposed into the fifth octave.)

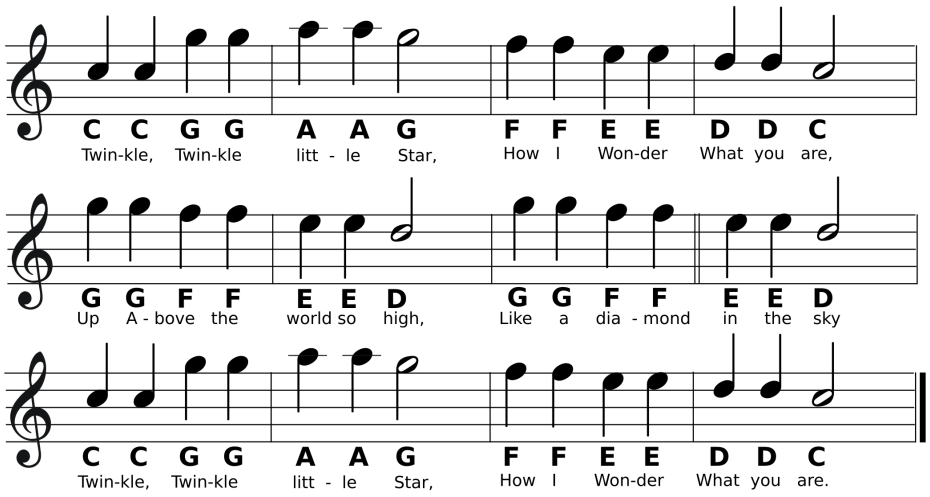


Figure 9: Twinkle-Twinkle Little Star. All notes are in the fifth octave.

- Add the circuit shown in figure 10.
- Find the variable `MaxAdjustSkips` and change the value to 20.
- Measure the output voltage of the divider with your hand about 1 foot above the photoresistor and with your hand less than an inch away. Put these values in for `MIN_LDR_VOLTS` and `MAX_LDR_VOLTS`.
- Have one person play notes with the remote and the other person make the notes "wobble" using the shadow of their hand. Show your instructor the final working circuit.

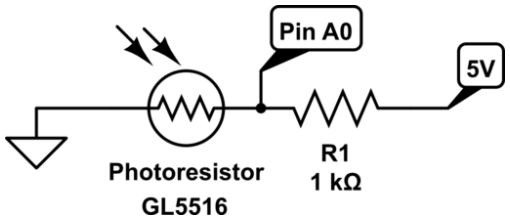


Figure 10: Light sensitive voltage divider