Physics 364, Fall 2014, Lab #22 **Name:** _____

(Arduino 3: Serial data and digital feedback — written by Zoey!)

Monday, November 17 (section 401); Tuesday, November 18 (section 402)

Course materials and schedule are at http://positron.hep.upenn.edu/p364

This file: http://positron.hep.upenn.edu/wja/p364/2014/files/lab22.pdf

Preamble: Damped Driven Oscillator and Serial Devices

Recall the equation of motion[1]:

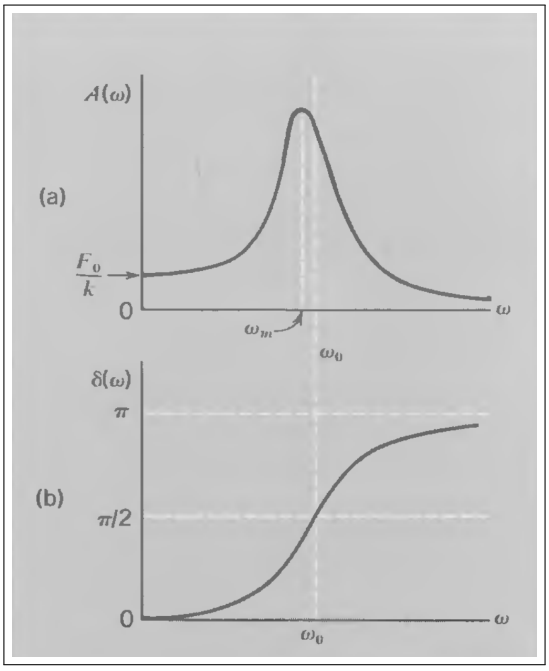
$$\frac{d^2x}{dt^2} + \frac{b}{m}\frac{dx}{dt} + \frac{k}{m}x = \frac{F_0}{m}\cos(\omega t)$$

or

$$\frac{d^2x}{dt^2} + \gamma\frac{dx}{dt} + \omega_0^2 x = \frac{F_0}{m}\cos(\omega t)$$

The driving force we will apply is coming from the interaction of the doorbell coil and a permanent magnet attached to the end of our spring. We'll not review the form of the force term here[2]. This setup is just a convenient way to create a feedback loop between our oscillator and the Arduino.

The response motion of a DDO is of the form $z(t) = A(\omega)\sin\big(\omega t - \delta(\omega)\big)$. Notice that the amplitude and phase, $A(\omega)$ and $\delta(\omega)$, respectively, are functions of the driving frequency of ω. We will use the Arduino to measure these properties of our DDO while varying the input frequency. Then we can build a search function to find resonance; that is, we will maximize the response amplitude and minimize $\delta - \frac{\pi}{2}$ in a closed feedback loop while varying the input ω.[3]



$$A(\omega) = \frac{F_0/m}{\left[(\omega_0^2 - \omega^2)^2 + (\gamma\omega)^2\right]^{1/2}}$$

$$\delta(\omega) = \frac{\gamma\omega}{\omega_0^2 - \omega^2}$$

There's a lot we could say about different types of data connections between devices. We will use SPI, a 4 wire connection interface. One disadvantage of SPI is that each device must also be powered independent of the data connection.

[1] This review of oscillators is covered in chapter 4 of French, A.P. *Vibrations and Waves.*

[2] For a review see Griffiths, D.J. *Introduction to Electrodynamics.*

[3] For practical reasons, we will only look to maximize the amplitude.

Part 1: Wire Up Accelerometer **Start Time:** _____

We will use the ADXL345 three axis accelerometer with a 4 wire SPI serial connection to read acceleration data from our damped driven oscillator to the Arduino and through a "serial" USB connection back to the computer. We also need two wires to power and ground the accelerometer. Connect the Arduino to the accelerometer as in the table below. The pin inputs on the Arduino are on the left of the table and breakout board is labeled as on the right. There will be two empty spots labeled INT.

Before connecting the Arduino to your computer[4]!

Arduino Pin	ADXL345 Pin
10	CS
11	SDA
12	SDO
13	SCL
3V3	VCC
Gnd	GND

The wires are soldered together the best I could do. It will be a little flimsy, but the range of motion should be relatively small. Plug in the Arduino to your computer. Use your web browser to download
http://positron.hep.upenn.edu/wja/p364/2014/files/arduino_adxl_blank.ino
and save it into your Arduino filder. Now from within the Arduino environment, open `arduino_adxl_blank.ino` and upload it to your Arduino. We'll make some modifications to it in the following sections.

[4]It probably wouldn't matter, but better to be safe.

Connect channel 1 of your oscilloscope (and trigger on it) to the CS wire. Connect channel 2 to the SCL wire. Describe what you see. What do you see when you connect channel 3 of the oscilloscope to SDA (which carries the serial data from Arduino to accelerometer — called MOSI in the reading)?

Each pin has a purpose. CS stands for chip select. This pin serves to tell the ADXL to start listening to the data channel. SCL is the clock channel which is how the ADXL recognizes the leading edge to clock data on. SDA and SDO are the write (MOSI) and read (MISO) channels respectively. VCC and GND have the usual meaning.

Part 2a: Program Arduino to Read Values **Start Time:** _____

Throughout the lab, in the code, look for commented lines with many hashes for where to fill in code.

The ADXL345 is recording data in a string of bytes on its local memory. You have to tell the Arduino to read those registers and pipe that data to the serial connection to your computer so you can see the measurements being made by the accelerometer. First, look in the unfinished function `readRegister()` and find the commented lines that show you where to fill in the reading logic. You may need to look at the ADXL345 data sheet:
http://positron.hep.upenn.edu/wja/p364/2014/files/ADXL345.pdf

Look for unfinished and commented lines. What is the code you used to read the byte string?

Now we have read the data into the Arduino's memory. Next we want to break the byte string in to x, y, and z coordinates and send it over the serial connection back to the computer. Note: The accelerometer records 10-bit data, so it is necessary to use 2 `char` bytes. The Arduino stores `int` as 16-bit numbers. Look in the main loop() function for commented lines.

What is the code you used to parse the byte string? Verify and upload when you finish. Check out the serial monitor.

Part 2b: Program Arduino to Handle Serial Data **Start Time:** _____

There's a line to remove from the Arduino code at the top of the `loop()` function before starting this part.

Right now the Arduino is just sending back data at the computer as fast as it can. Open the serial connection in the Arduino IDE (it's the magnifying glass icon on the upper right of the window) to see what data you are getting. We really only want to get data from the Arduino when we ask for it.

The `serialEvent()` function, if it is defined, runs between each iteration of the `loop()` function. `serialEvent()` listens for activity on the serial port and then executes before the loop function is set. This way we can tell the Arduino to set a flag and read data whenever there is a particular `serialEvent()`. Define `serialEvent()` to listen for a particular character (I chose "G" for get). If it sees the particular character you want on the serial connection (use the function `Serial.read()`) tell the Arduino to `readRegister()` from the ADXL345 in the values data.

Upload your new script and open the serial connection inspector. You shouldn't get any data unless you explicitly ask for it using the send data box at the top of the serial data window.

Part 3: Processing graphics **Start Time:** _____

Now we will use the Processing language (nearly identical to Arduino programming language) to visualize our data. Processing has a function called `draw()` that runs repeatedly like the `loop()` function in Arduino. Think about it this way: every frame we want to draw a grid like the background of our oscilloscope and we want to draw our data to the screen as well. Start from this partially-written Processing sketch, which you should download, save, and then open from the Processing application:
http://positron.hep.upenn.edu/wja/p364/2014/files/processing_adxl_blank.pde

Under the comment that says `//Graphics & drawing data to screen` there needs to be some code that draws the grid and data to the screen. This part is optional, so you can find solution code at the bottom of the file, or you can try it yourself. The `for` loop goes over every column of the display window. It should draw the grid and the data at every point. And it should advance our data buffer after every point is drawn. Finally after the loop we append the latest data point to the buffer.

Write some code at the end of the `draw()` function that will compute and print out the amplitude averaged over the oscillations in the data buffer.

Part 4: Reading Serial Data in Processing **Start Time:** _____

Here we want to read the serial data being written to the line by the Arduino after we request it at the top of the `draw()` function. Just take a look at the function. No need to do anything else here.

Part 5: Drive the Spring **Start Time:** _____

First just force the spring manually. You should be able to see the oscillations in your Processing window. Next try driving the oscillations with the function generator and the doorbell coil. Tune the frequency to get the largest amplitude you can. It will probably be something greater than 1Hz but less than 10Hz.

Part 5$\frac{1}{2}$: Analyze and Build Feedback **Start Time:** _____

If you've made it this far with time to spare, try to build the feedback mechanism into your Arduino so you don't have to control the function generator by hand. There are several steps to this part.

First you will need to amplify the current to drive the doorbell coil. Try a MOSFET switch. Next code a pin on the Arduino to toggle on and off at variable frequency between 1Hz and 10 Hz. 100mHz precision should be ok. You will also need a function on the Arduino that measures the amplitude and decides to increase or decrease the driving frequency depending on the Amplitude response. This will be tricky. (In fact Zoey hasn't done this part yet himself! So if you make it work, we will be **very** impressed!)

Optional 6: reaction timer (same as Lab20 optional6)

Using one push-button input (with connection to +5 V and pull-down resistor to ground) and one LED (with current-limiting series resistor), make a **reaction timer** that measures how quickly your finger responds to a visual stimulus from the Arduino.

The Arduino should first turn on the LED, then wait for you to press a button. Once you press the button, the Arduino should turn off the LED, then delay a length of time 1000 ms $< \Delta t <$ 5000 ms, then turn on the LED. The Arduino then measures how long it takes before you respond by pressing the button. As soon as you press the button, the Arduino turns the LED off, then reports (via `Serial.print()` to the serial console) the number of milliseconds that it took you to react. After another brief delay, the Arduino turns the LED back on and returns to the initial state.

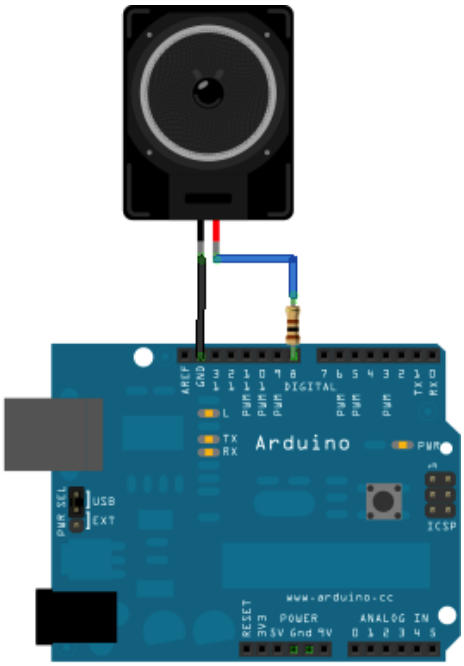
If you do this exercise, you don't need to write anything down, but please call us over so that you can show off your results!

Optional 7: musical tones (same as Lab20 optional7)

First make a sketch whose `loop()` function first outputs `HIGH` on pin 8, then waits 1.136 ms by calling `delayMicroseconds(1136)`, then outputs `LOW` on pin 8, then once again calls `delayMicroseconds(1136)`. Check with your oscilloscope that the result is a 440 Hz square wave on pin 8.

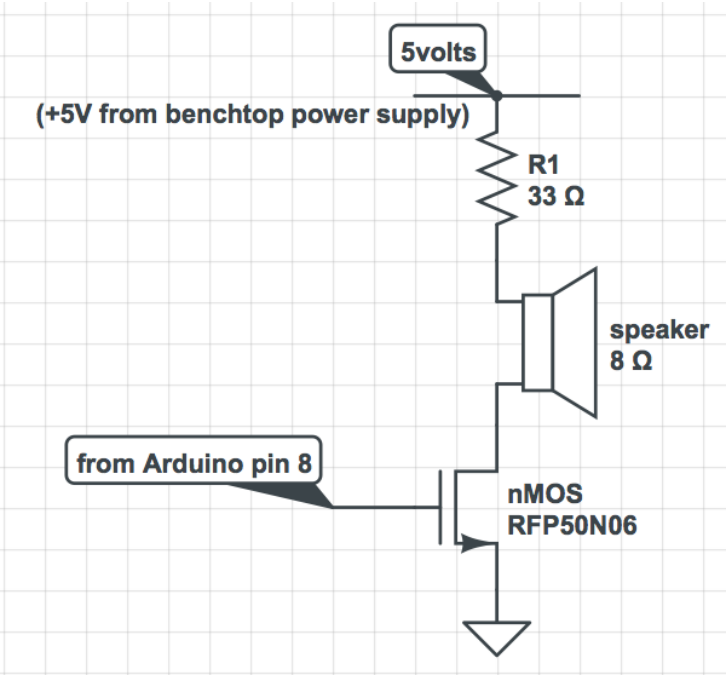
Now notice that the same result can be achieved much more easily by calling the built-in function `tone(pin, frequency)` where in this case `pin` = 8 and `frequency` = 440. You can also call `tone(pin, frequency, duration)` where `duration` is in milliseconds.

Next, connect a 100 Ω resistor and a speaker in series from pin 8 to ground, as shown below. You should hear (quietly) the **A** note above middle C.



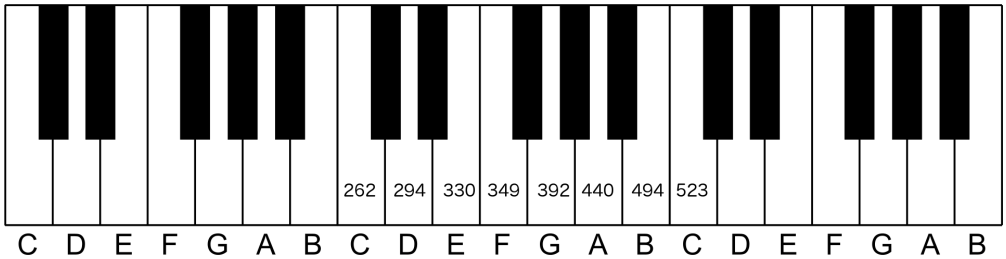
(Continued on next page.)

If you really want to make enough noise to annoy your neighbors, you can try using a MOSFET switch to boost the power supplied to the speaker! Here is a circuit that I (Bill) tried out in the lab the other day:

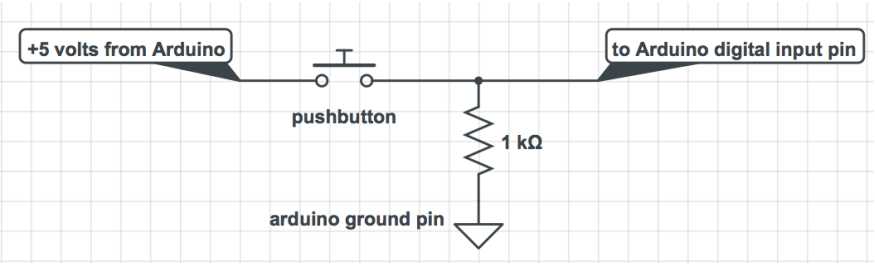


Whether you go with the quiet or the noisy version, here are two more challenges you can add to this exercise:

First, try playing a major scale, starting from middle C (262 Hz). The notes C, D, E, F, G, A, B, C that you need to play have frequencies 262, 294, 330, 349, 392, 440, 494, 523 Hz, as indicated on the keyboard below. Notice that the frequency ratio between octaves is 2, between adjacent keys ("half step," e.g. E to F) is $2^{1/12} \approx 1.06$, and between two keys having one key in between ("whole step," e.g. C to D) is $2^{1/6} \approx 1.12$.

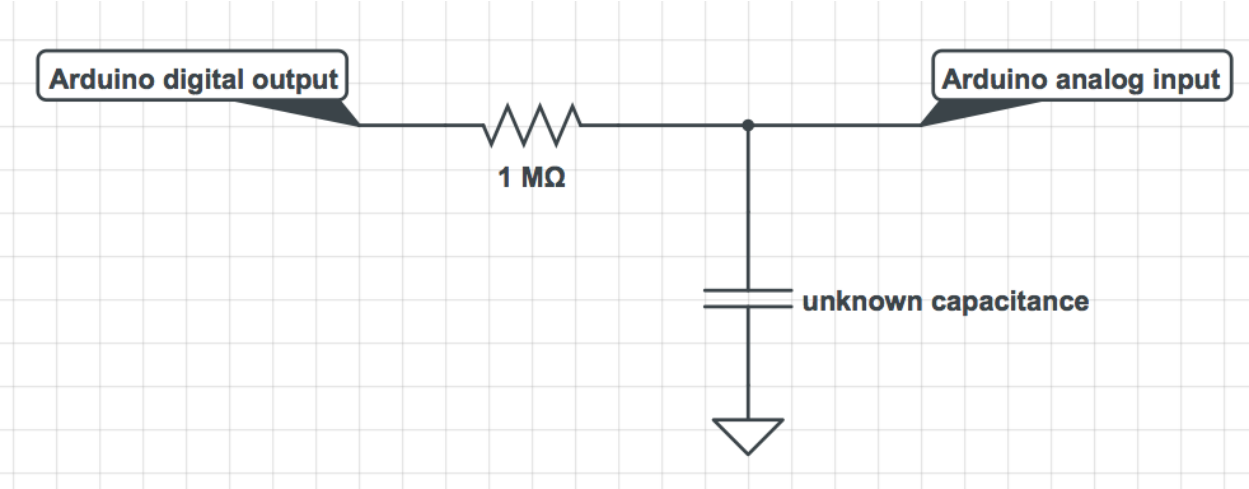


Second, if you're feeling ambitious, try making a simple 8-button musical keyboard, using 8 copies of this circuit to drive 8 Arduino digital input pins:



Optional 8: capacitance meter (new!)

Here is a chance to apply the Arduino's own built-in analog-to-digital converter, in order to build something potentially useful — a *capacitance meter.* The idea is this: first, the Arduino drives a digital output to 0 V, to discharge the capacitor. Then it monitors an analog input until the voltage reads $\approx$ 0 V, so that it knows the capacitor is fully discharged. Then it records the present time, using e.g. `int time0 = millis();` , drives the digital output to +5 V, and monitors the analog input until V_{in} exceeds (5 V)$(1 - e^{-1}) \approx 3.1606$ V. Then it records the stop time, using e.g. `int time1 = millis();` . The unknown capacitance, determined from the known resistance R and known decay time $\Delta t = RC$, can then be displayed on the serial monitor, in nanofarads. Once the meter is verified using a known capacitor, try it using several "unknown" capacitors.

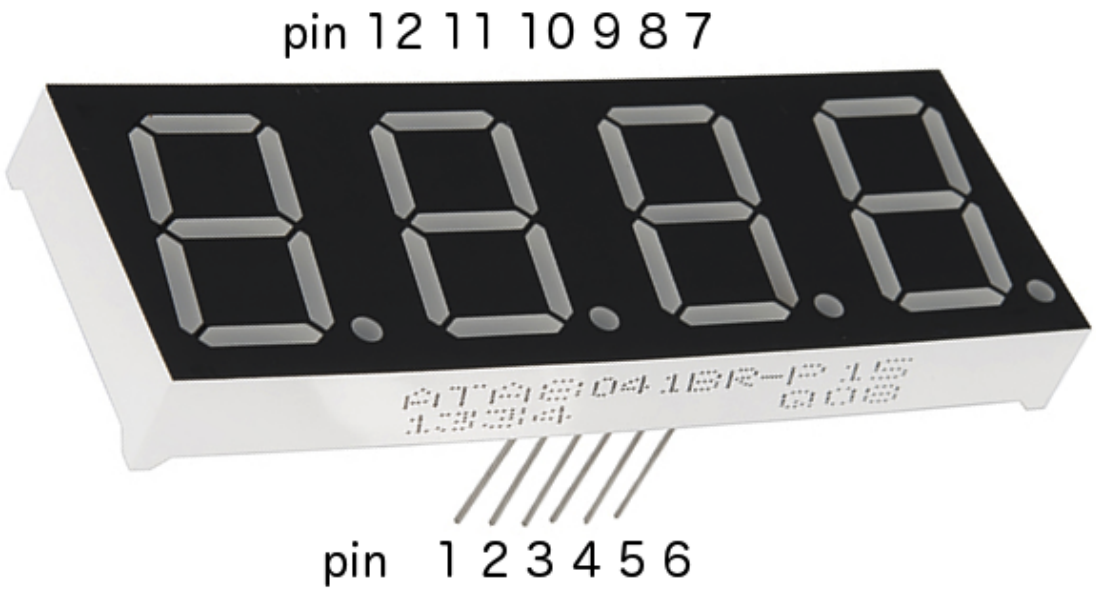


First, learn to use the Arduino's `analogRead()` function, from this example:
http://arduino.cc/en/Tutorial/AnalogReadSerial

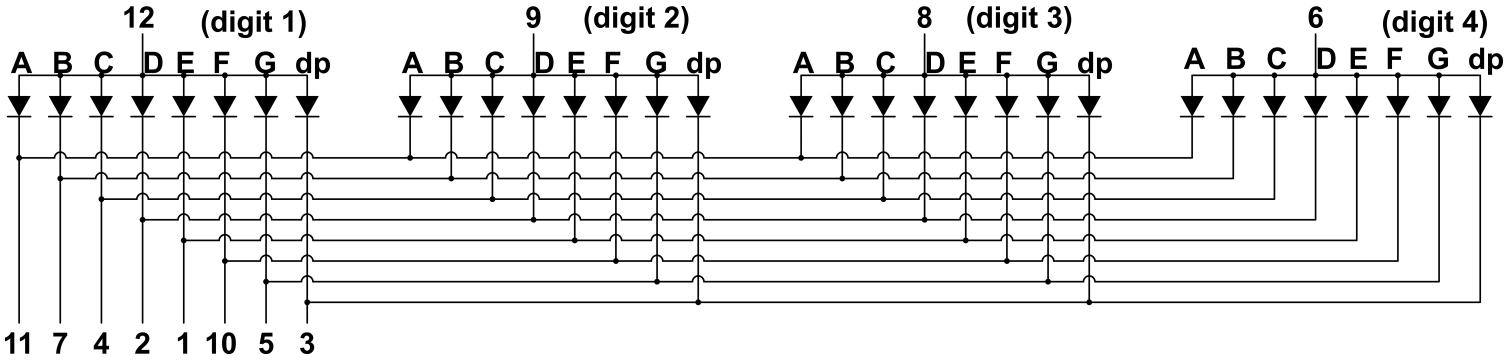
Then wire up the above circuit and code your sketch. Once you get it to work, please be sure to show it off to us! When you're done, save your sketch, because you may have a chance to build upon it later.

Optional 9: 7-segment LED display (new!)

Seven-segment LED displays are widely used as a simple way to display digits 0-9 and sometimes also letters A-F. We will use 7-segment displays during the FPGA labs at the end of the course. This device offers you a chance to get the Aruino to display numbers in human-readable form without the help of the PC serial-port monitor. The display consists of 4 digits, each of which is composed of 7 separate LED "segments" (i.e. line segments), plus a decimal point. The display is conveniently breadboard-compatible. The pins, as usual, are numbered from lower-left, increasing in the counterclockwise direction.



One would think that $8 \times 4 = 32$ separate pins (plus at least one common power or ground pin) would be needed to control 4 digits, each of which consists of 7 segments plus DP. But the manufacturers of these devices use a clever trick, shown below, to reduce the total number of wires needed to just $8 + 4 = 12$. Essentially, the anodes (+ sides) of the 8 LEDs for each digit all share a common pin, while the cathodes (− sides) of each corresponding segment (e.g. the bottom segment) for all 4 digits share a common pin. So pin 12 connects to the "+" sides of all eight LEDs for digit 1, while pin 11 connects to the "−" sides of the top segments for all four digits. So pins 12,9,8,6 select which digit is illuminated, while the other pins select which segments of the selected digit are illuminated.


12
(digit 1)
9
(digit 2)
8
(digit 3)
6
(digit 4)
A B C D E F G dp
11 7 4 2 1 10 5 3

The right figure below indicates the naming scheme (A–G) for the 7 segments that compose each digit. The decimal point is called "dp." The four digits are numbered 1–4 from left to right. To make the lower-left segment (segment E) of the left digit (digit 1) light up, you need to connect pin 1 ("E cathodes") to ground, and pin 12 ("digit 1 anodes") **through a** 1 kΩ **series resistor** to +5 V. **Don't forget the series resistor,** as the maximum allowed current per LED segment is 20 mA, while the voltage drop across the LED (when illuminated) is about 1.9 V.

To make digit 2 display the numeral "7" you would connect pins 11,7,4 (segments A,B,C) to ground and pin 9 (digit 2) **through a** 1 kΩ **series resistor** to +5 V.

pin	function
1	segment E cathodes
2	segment D cathodes
3	decimal point cathodes
4	segment C cathodes
5	segment G cathodes
6	digit 4 anodes
7	segment B cathodes
8	digit 3 anodes
9	digit 2 anodes
10	segment F cathodes
11	segment A cathodes
12	digit 1 anodes

https://www.sparkfun.com/products/retired/12098

Let's start by teaching the Arduino to count from 0 to 9 using only digit 1. Connect pin 12 **via a** 1 kΩ **series resistor** to the Arduino's +5 V pin. Then connect pins 11,7,4,2,1,10,5 to seven of the Arduino's available digital output pins. To light a given segment, set that output `LOW`; to dim the segment, set that output `HIGH`. To light the correct LED pattern for each digit 0–9, figure out which segments need to light up, and drive the corresponding outputs `LOW`, and the other outputs `HIGH`. The next page will be blank, in case you need space to think.

Next, try driving the anode pins for digits 1,2,3,4 (pins 12,9,8,6) from four additional Arduino digital outputs (rather than the +5 V pin), **using a separate** 1 kΩ **series resistor** for each of the four anode pins. Try counting with digit 1, then counting with digit 2, then counting with digit 3, then digit 4.

Next, try counting from 0000 to 9999, displaying the 1000's place on digit 1, the 100's place on digit 2, the 10's place on digit 3, and the 1's place on digit 4. You can use the division operator, `/`, and the modulus (remainder) operator, `%`, to extract the digits:

```
counter = counter + 1;
int digit1 = (counter/1000) % 10;
int digit2 = (counter/100) % 10;
int digit3 = (counter/10) % 10;
int digit4 = counter % 10;
```

To display a four-digit number, first display `digit1` on digit 1, for a few hundred milliseconds, then display `digit2` on digit 2, for a few hundred milliseconds, ..., then display `digit4` on `digit4`. During each digit's display, you drive only that one digit's anode `HIGH` (driving the other anodes `LOW`), and you drive the corresponding cathodes `LOW` to light the desired pattern of segments on that digit.

It may look silly at first to display a four-digit number by lighting one digit at a time for a few hundred ms. But now change the time for which each digit is displayed to just a few tens of ms instead of a few hundreds. Your eye will be fooled into thinking that all four digits are illuminated at once, and you will see a four-digit number!

If you manage to get all of this working (!), try modifying either your capacitance meter or your reaction timer to display its output as a four-digit number on the LED display rather than on the PC's serial monitor.

If you achieve the monumental task of making this four-digit display work, you should **definitely** show it off to us!

Finally, there is some small possibility that the Arduino Uno board's power supply may not be capable of supplying the necessary current to drive this LED display. We doubt that this is true, as Jose succeeded yesterday at displaying one digit with the Arduino. Nevertheless, if you do run into this problem, there is an easy work-around using a MOSFET switch to power each anode pin from the external +5 V benchtop supply **with a** 1 kΩ **resistor in series with each anode pin**. It is the same trick that we used to make the speaker louder in optional exercise 7.