Lab 4: PCB design: Schematic capture and layout in KiCad

In this lab you will design a printed circuit board implementation of the R-2R DAC. By the end of the lab, you will have a set of design files that can be sent to a PCB manufacturer for fabrication.

The general flow of computer-aided circuit design is to first enter a specification of your circuit into the computer, then create a design for a physical implementation of the circuit. In the case of PCB design with KiCad, circuits are specified as arrangements/connections between components which can can be soldered to a PCB. As components are usually items that are purchased from a supplier (Texas Instruments, Analog Devices, etc.), their symbolic representations in the schematic editor and their physical representations in the layout editor (their Footprints) are considered to be set in advance of a design. Of course, the symbols and footprints must be defined somewhere, which is why KiCad includes symbol and footprint editors; you can think of these editors as being a place to enter information from a component's datasheet into KiCad, while the schematic and layout editors are where you create your own design.

One of the key functions that KiCad performs is linking the conceptual representation of the circuit (the schematic) to the physical implementation (the PCB layout). However KiCad will only be as accurate in this task as you are, which is why it was so important to match the symbol pin numbers to the footprint pad numbers. KiCad uses this link to show you what connections need to be made in the layout, and to automatically verify that your PCB design is correct.

4.1 Schematic Editor

Schematic capture refers to the process of entering a circuit design into a program via a graphic interface, i.e. drawing a schematic using a program that understands what the schematic means (alternatives could be drawing a schematic in, say, PowerPoint, which knows not what you are doing; or by entering a schematic as a net list, which is convenient for a computer but not for us). When you draw a schematic in KiCad, KiCad understands that when you draw a wire between two or more pins (i.e., the pins of a symbol) that those pins should all be connected together on the same "net" or "node" (two words that mean the same thing in this case). The upshot of this is that KiCad now knows that when you go to design a PCB for your circuit, that the corresponding pads on the PCB need to all be shorted together using the copper layers and vias (conductive pathways between different layers of copper in the PCB) that the PCB manufacturer can produce for you. Thus it is the human's job to ensure that the schematic is right, while it is the computer's job to check that the PCB layout matches the schematic. If the schematic is wrong, KiCad will only tell you that your PCB is right if it is in fact wrong the same way that your schematic is wrong.

4.2 Schematic Editor walk-through

1. Create a new project by "File > New Project..." or pressing Ctrl+N/CMD+N from the KiCAD Start screen. Save it under the folder of your choosing
2. Press Schematic Editor. If a plugin popup appears, click yes to automatically update the plugins.
3. You will now see the schematic editor. A labelled image of the schematic editor can be seen in 38.

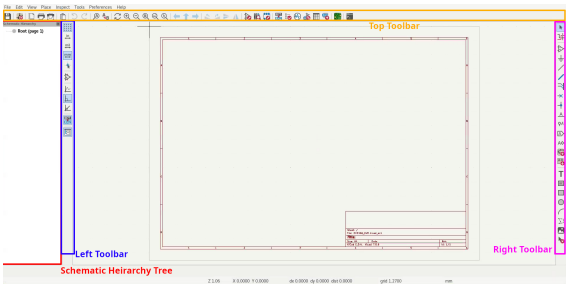

Figure 38: Schematic editor with labels

4. Start building your schematic by adding a resistor from the device library. This is done by pressing the "Add a symbol" button on the right toolbar. You can see this walked through in Figure 39. When you select the component that you want to place, you can also select which footprint you want to associate with the symbol. You can start placing components with no footprints attached, since we will guide you through footprint selection later.

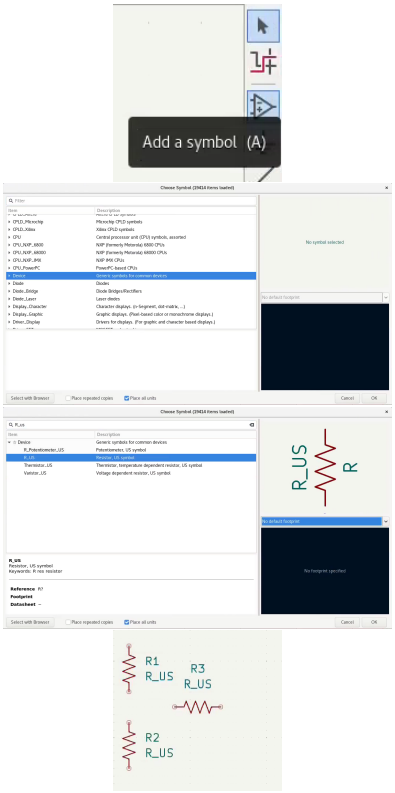

Figure 39: Order of placing a Resistor symbol from the symbol browser

Each component like the resistor that you just placed has a set of properties that can be viewed and edited in a dialog that can be accessed by pressing "e" while you have the part selected (after you have placed it in the schematic). The three most important fields are available at the top: Reference, Value, and Footprint. The Reference field holds a designator (i.e. R1, R2, etc.) that uniquely identifies each component in the schematic. KiCad 7 should automatically populate this field, but you can change it. If it isn't populated, we'll fix it later. The Value field is more human-oriented, and annotates each component with useful information. For resistors, capacitors, and inductors, it is used to hold the resistance, capacitance, or inductance of the part. For other parts such as integrated circuits, it usually holds the part number (i.e., "MCP602"). The value field can also be directly edited by pressing "v" while the component is selected. As you go, fill out the value for each resistor that you place. The resistor we selected will default to "R_US", as seen in Figure 40.

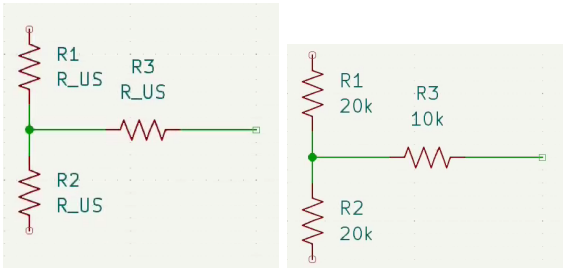

Figure 40: Making a R2R as a schematic

5. Keep adding resistors to build a schematic for the 8-bit R-2R DAC. Components can be copied and pasted, which can be convenient since any footprint and value selections will be copied as well, while the reference should automatically increment. You can even copy and paste each whole "stage" of the R-2R DAC. The pins of the resistors are wired together using "add a wire" function, accessible on the right toolbar near the top. It should look something like Figure 41 when you are done.

Figure 41: Making a R2R as a schematic

6. Net labels are an extremely useful way to connect sections of the circuit without drawing wires everywhere. All wires that are given the exact same label will be understood as being connected. In Figure 42, you can see how labels are applied. While labels can greatly help the readability of your schematic, they should not be over used; in general, you need to decide for yourself whether labels or direct wires will make the connections in your circuit easiest to follow. KiCad has several types of labels for different scenarios, but for this project make sure to stick to "Net Labels" specifically.

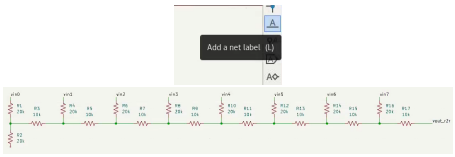

Figure 42: R2R Labelled

7. Add the amplifier we created in the previous sections of this lab to the schematic. This addition is shown in Figure 43. You can find your part by searching MCP602 in the Add symbol dialog, keep the checkbox of "Place all units checked" for the sake of this lab. Since we will be using the amplifier as a voltage buffer, the positive input of one of the amplifiers in the MCP602 should be connected to the output of the R-2R DAC, and the output of the amplifier connected to the negative input, for negative feedback. The second amplifier in the MCP602 chip can be left unconnected, since we will not be using it for anything.

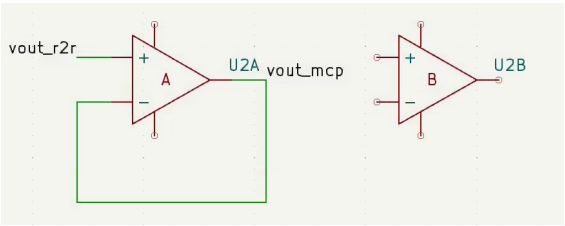

Figure 43: MCP602 Amplifier Added

8. Add the amplifier LTC1152. This addition is similar to the MCP602. The default symbol from the symbol libraries works well. You can see this component added in Figure 44.

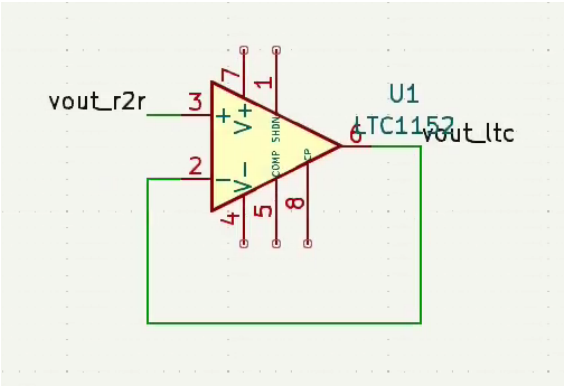

Figure 44: LTC1152 Amplifier Added

9. Power supplies (in our case, +5V and ground) are usually included in schematics using a special set of "power symbols". These symbols act somewhat like net labels, in that all wires connected to the same type of power symbol will be connected together. Add power symbols by pressing "P". You can see a step by step image walkthrough in Figure 45. You will see multiple different GND power symbols; it doesn't matter which is chosen as long as it is consistent between each of the placed symbols (don't mix and match, since they won't connect). Remember to connect power and ground everywhere it is needed, and ensure that the 5V power net is connected to the shutdown pin on the LTC1152 so that the device is powered up.

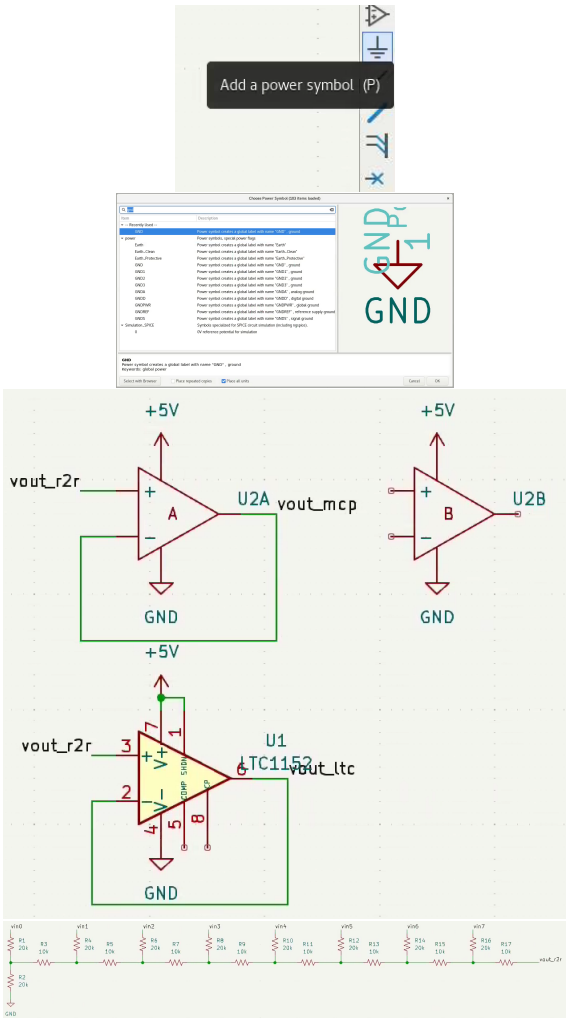

Figure 45: Power symbols added

10. We will attempt to place "pin header" connections on our PCB so that it can directly plug in to / stack on top of the Arduino. Pin headers are one of the easiest and most standard connections that are used in non-consumer electronics - they're just pins, almost always set 0.1 inches apart, that can connect to other boards or to standard jumper wires. Even though in some sense they are just wires, they are still components, since they are items/objects that will be soldered to the PCB.
    The Arduino pin headers are arranged in 1 group of 6, 2 groups of 8, and 1 group of 10 that should be kept separate because the offset between these headers are not on a constant grid (An engineering oversight that will become more apparent later in the PCB design). To place it for easy reference (because of the two groups of 8), place the header symbols (search Conn_01xNN, replacing NN with 06, 08, or 10 as needed) such that they are organized as they are on the Arduino. See Figure 46 for an example of this. This positioning isn't technically necessary in the schematic, but is recommended to keep the schematic more organized.

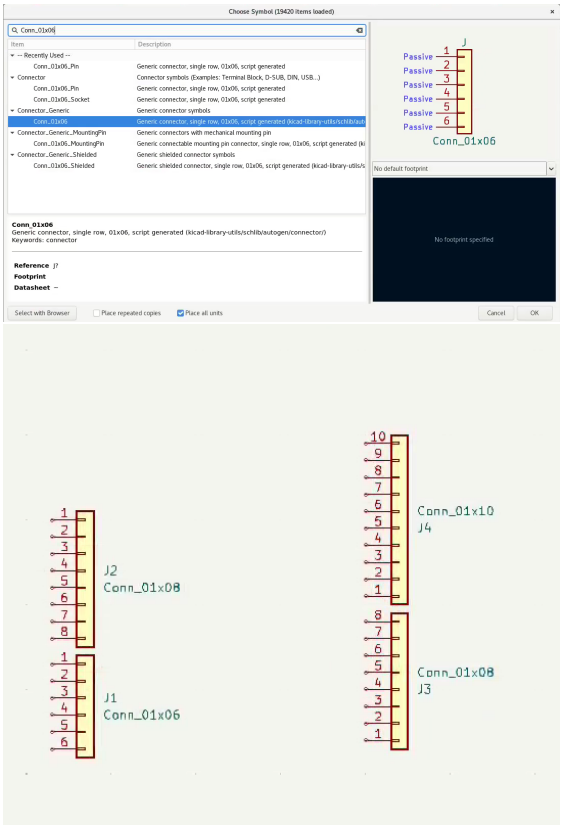

Figure 46: Placing header connectors

11. Connect the header pins to the circuit elements, keeping in mind that in some sense this represents connecting to the Arduino headers. Arduino digital outputs D2 to D9 should be used as the inputs of the R2R DAC (the figure shows D0 through D7 but that will cause the serial monitor to stop working when you test your PCB, so offset the pins so that D2 through D9 are connected). Additionally connect the output of the R2R DAC to A0, the output of the LTC to A1, and the output of the MCP to A2. And connect the GND and 5V pins so that the Arduino shares its power with your PCB, as seen in Figure 47. You will notice that the right connectors have been reflected so that their numbers increase in the upwards direction. This is to reflect that pins on components increase in the counterclockwise direction and will help orient the parts on the PCB layout. These are placed oriented downward to mount directly into the Arduino.

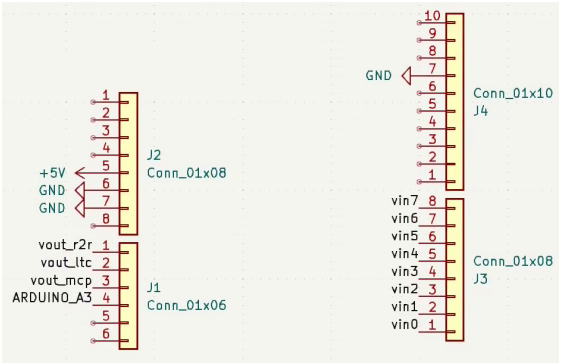

Figure 47: Header connectors labelled

12. Add an additional header with 5 positions (GND, A0/R2R, A1/LTC, A2/MCP, and A3) so that we can connect our circuit to a breadboard at the same time it is plugged in to an Arduino. The connections to make can be seen in Figure 48.

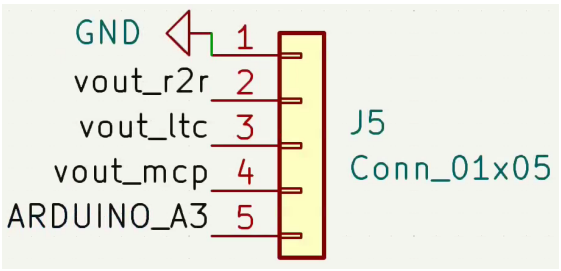

Figure 48: Header connectors for the top of the board

13. Ensure that the circuit is fully annotated. This is not necessary in more modern versions of KiCAD, so it is only necessary if your schematic designators are in the format of R?? instead of, for instance, R1. This is found under "Tools > Annotate Schematic" and press "Annotate".

14. Attach footprints to every component in the circuit. For the connectors and resistors, add the filters to restrict it by the filter from the symbol, filtered by the selected library, and filter by pin count. Filtering by pin count is not always helpful and sometimes inhibits footprint selection, but for fully populated symbols, like the ones in this lab, it is helpful to have on. For the connectors, navigate to the library "Connector_PinHeader_2.54mm" and chose the through hole (the footprint NOT labelled SMD) vertical pinheader that matches the pin count of the connector (There is an error in the figure as originally there was a 4 position header for the 5 position top header, so choose the relevant 5 position header, which you will see in the following figures but it is just one additional connection that will be made and all the positioning and other connection information is accurate). A good resistor footprint is "R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_" which leaves a good amount of room for comfortable soldering of 1/4W resistors while not oversizing the layout (The most important part is the P10.16mm_Horizontal, which refers to the pitch of the pads and the orientation of mounting). Lastly, use the stock footprint in the Package_SO to add a footprint to the LTC1152 IC. You may note that the MCP602 does not have a value here, which can be added in the symbol. These are seen in Figure 49.

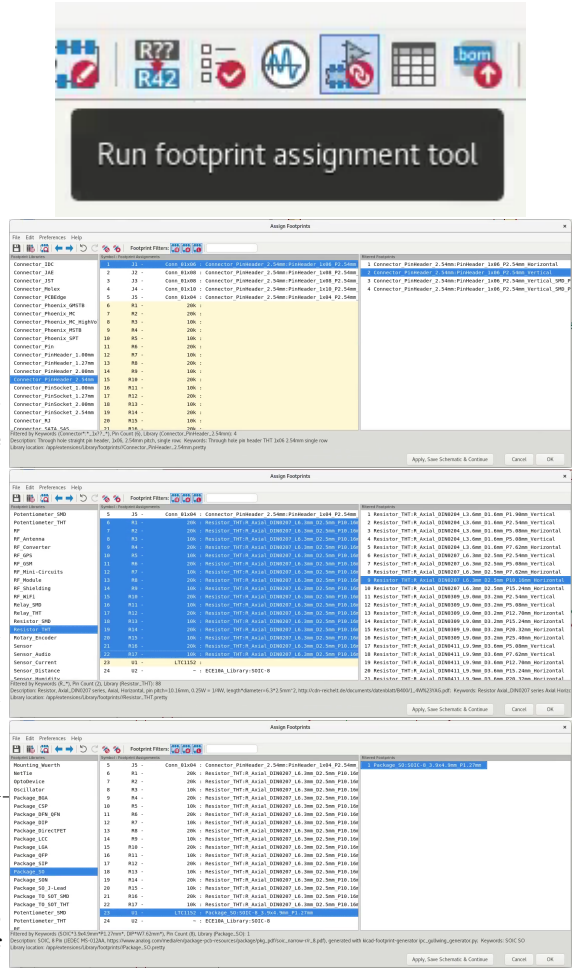

Figure 49: Connecting footprints to symbols in the schematic

4.3 PCB Layout

Printed circuit boards (PCBs) are made in a layered fashion. Layers of copper are patterned to create wires, and different layers of copper are separated and insulated from each other using some other material, usually FR-4 (a fiberglass-like material). Where layers of copper overlap, the different layers can be shorted

together using "vias" - small plated holes in the PCB. More complex PCBs can have many layers of copper sandwiched between insulating layers, but simpler PCBs - including the ones you will be designing - have a single layer of insulating FR-4, with one copper layer on each side. These "external" copper layers are then insulated by "solder masks", which are thin polymer layers that selectively protect and insulate the copper layer. The solder mask is present almost everywhere on a PCB, except for locations where components need to be soldered to the copper layer - hence the name solder mask. On top of the solder mask, there is an ink layer called the silk screen which allows for annotating components and placing any other labels on the PCB.

Since a PCB is created by stacking up patterned layers of material, it is only natural that PCB design software is also organized around "layers". Each layer is a binary vector drawing, representing whether or not the particular material layer is present at a particular point on the PCB. Most layers are defined positively (i.e, copper will be placed where there is some drawing on the copper layer), except the soldermask layers, which are defined negatively - you draw where it will be absent, instead of where it will be present. Since all the layers we have mentioned are present on both sides of the PCB, there are two copies of most layers - "F" for front side layers, and "B" for back side layers.

There are a number of additional layers in PCB layout software which are not present in the material "stackup" (as the set of material layers is often called). Some that you have encountered during footprint design are the fabrication layers and the courtyard layers, which are intended to aid in arranging components on the PCB, but are not actually manufactured. This particular Author finds the fabrication layer to be cluttersome, and usually hides it when doing layout. One last layer that is critically important is the board outline, or edgecut, layer. You will draw a polygon in this layer to indicate what the extent of your PCB design is.

In addition to the layer system, the PCB editor also has two kinds of objects. The first is one that you are already familiar with: footprints. Each footprint, which describes the pattern of contacts on a PCB necessary to solder a particular part to the PCB, is essentially a group of vector drawing elements across various layers of the PCB. The pads of the footprint include both patches of copper for soldering the component too, and openings in the soldermask so that the copper is actually exposed - since soldermask openings are handled by the footprint objects, you will rarely (and certainly not in this lab) have to explicitly use the soldermask layer. The second type of object is vias, which create connections between the top and bottom copper layers. Vias place material on the various copper layers, but also contain information for an additional PCB fabrication step that is not so layer-like: x-y coordinates where holes will be drilled in the PCB. When you get to exporting the design files for your PCB, these coordinates will be exported in a different format than all the other layers.

4.4 Layout Editor walk-through

1. The PCB Editor is opened using the PCB Editor button, found either in the top toolbar of the Schematic editor on the right side or in the main set of buttons in KiCad's start window.

2. There are several contextual options associated with each PCB layout that can be edited or viewed at any time, such as the number of copper layers (default: 2) and the "Design rules". While all of the defaults are suitable for the current project, it is good to know what they are and where to find them. "Design rules" are a set of dimensional specifications for how accurately the PCB can be made, and put a lower bound on dimensions such as the width of a wire, or the spacing between two wires. These limit essentially say, "if you design a wire smaller than this, it might not be printed correctly - you PCB might not work". The layout editor enforces some design rules as you work, and also has a function to automatically check for design rule violations across your entire PCB once you are done. Design Rules can be accessed through a button on the left of the top toolbar. For this tutorial, the defaults work fine enough because they are conservative relative to what most board houses are capable of (i.e. most PCB manufacturers can make smaller features than what KiCad sets as the minimum). If you need to change the design rules in the future, which is fairly frequently the case, find the PCB manufacturer's website and find the page for the design-rule check (DRC) for the process in question, and put the information of that page into the DRC constraints dialog. You can see the defaults in Figure 50.

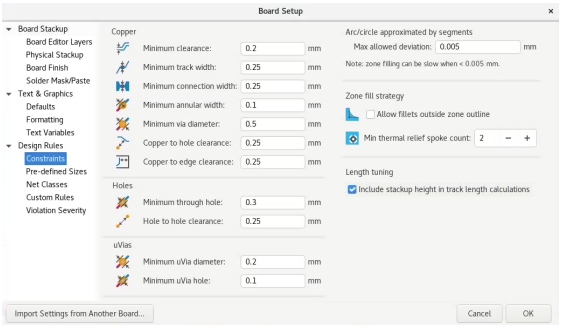

Figure 50: DRC Dialog

3. Start by importing/synchronizing the footprint netlist into the layout. You can see how to do this in Figure 51. This step transfers the information contained in the schematic to the layout editor, and must be repeated each time you edit the schematic - changes will not automatically be reflected in the layout otherwise. You might get warnings or errors in the "Changes to be Applied" log if you forgot to assign footprints to all symbols, or if there are any other issues with the schematic. Even if you have warnings or errors you can still continue and click to place the imported components in the PCB view for initial placement. You will probably want to address the warnings or errors later, if you have any.

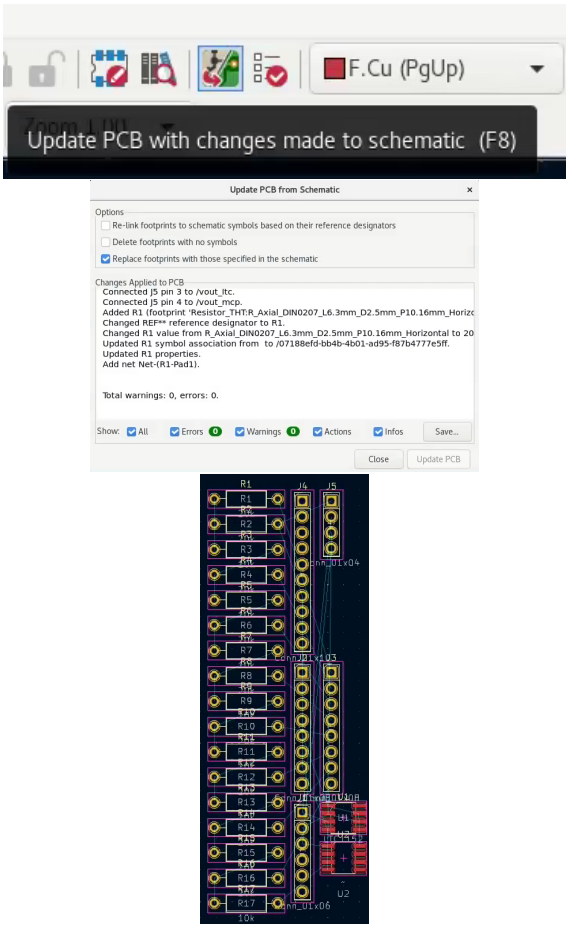

Figure 51: Importing footprints into the PCB design

4. Since we have pin headers that we want to directly mate to the Arduino, the position of these parts are fixed by external requirements so it is a good idea to place these first. This is where some annoyance from the Arduino's design comes in to play. Because the engineers responsible for the design of the Arduino built it working (mostly) on a thou ($\frac{1}{1000}$ of an inch) grid, make sure to switch to an imperial grid spacing for this part, ideally 100 thou. Additionally, for now select the "Use Mil" Button on the left toolbar. It would have been nice if all the pin headers were on a 0.1 inch grid, but they're not, not quite.
   To get the positions just right, you can use the following procedure. You will use the analog input header as the positional reference, and place the other three headers by first overlapping them with the analog header and then using the "Positioning Tools > Move Exactly..." tool, accessible via right clicking on a selected part to create the correct spacing between each header. The correct center-to-center offsets are shown in figure 52. Note that the Y deltas are negative, since KiCad defines zero as the upper left corner of the page.

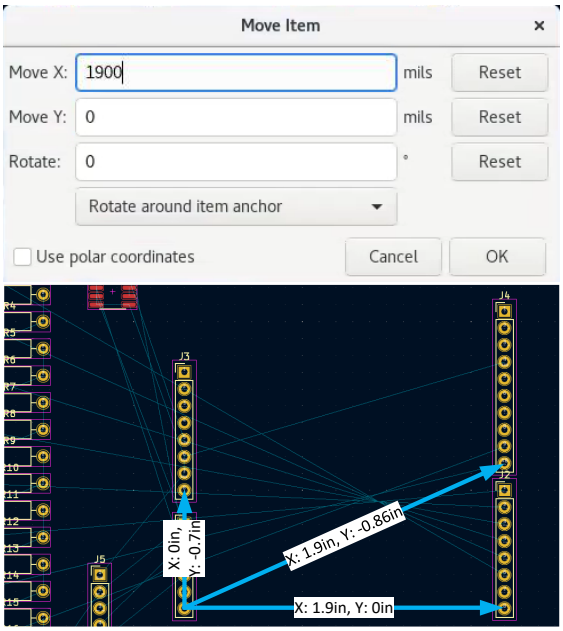

Figure 52: Positioning the headers relative to one another

5. After the pin headers have been placed, the remaining circuit components need to be arranged. They should all be within the region outlined by the pin headers; if you place components outside of the Arduino-shaped bounding area, your PCB will be larger, and perhaps more expensive, than it needs to be. A good initial guideline is to arrange the components to minimize the bounds of the layout and "airwire" crossovers. The airwires are the lines between footprint pads, and indicate how the pads need to be connected in order for the layout to match the schematic.
   Each component can be on either the front or back of the PCB, and can be switched between the two using the flip command ("F" key press). For dense designs, the selection of front/back is an important degree of flexibility when choosing placement. That being said, this author recommends keeping as many parts on the front as possible: from a debugging/test perspective, having everything on the front makes it easier to see how a circuit is organized just by looking at a PCB, and allows for directly probing the contacts of surface-mount parts to figure out what's wrong with a design. For this project, the only parts that should be flipped are the pin headers for connecting to the Arduino. Although this shouldn't affect the PCB much at all since the pin header footprints are symmetric, it is good bookkeeping and makes KiCad's 3-D PCB rendering more accurate - the rendered pin headers will show up on the side you will be soldering them to.

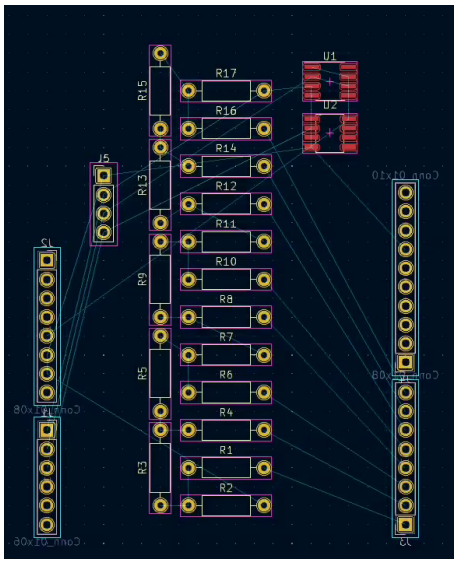

Figure 53: Approximate final position of components

6. Connect the components that are not the power nets (GND and +5V). Pressing X will allow you to interactively route, pressing "-" and "+" will automatically place a via to get from the top layer to the bottom without interrupting routing. You can see an example of this in Figure 54. You should be able to route everything on the top layer but it may be a bit cluttered, where as if you route some of the routes on the bottom, you can make a much cleaner PCB.

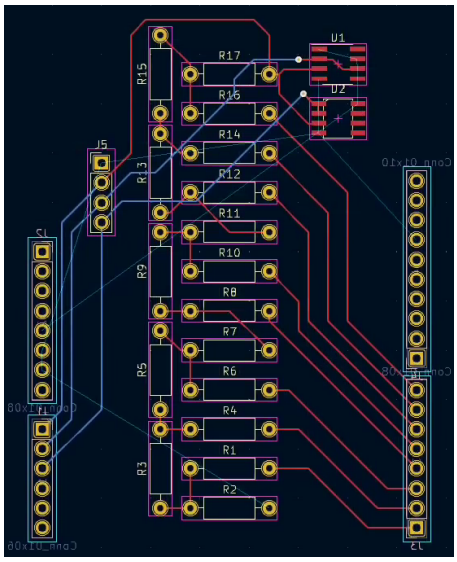

Figure 54: Routing the core components of the PCB

7. Add the board cut, which defines the extent of the PCB. Select the "Edge.Cuts" layer in the layer panel on the right and select the line tool (Note this is different than the wire tool used to draw the routing). Draw a rectangle around where you want the edges of your board to be. It can be chamfered if you wish. See Figure 55 for this.

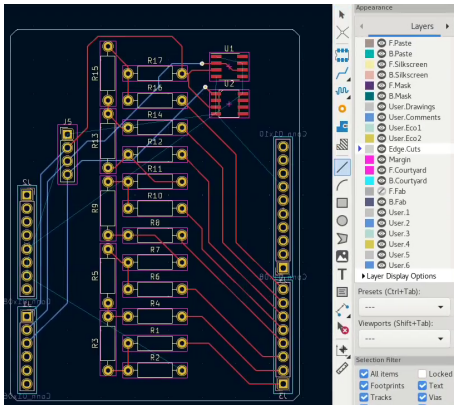

Figure 55: Adding the edge cuts to the PCB

8. Although we could have routed the ground and +5V power nets the same way as the other signals, it is better practice to use power planes/fills - this both reduces the resistance in your power supply net, and reduces parasitic inductance (which really isn't important here, though). Fills are specified as a region and a net, along with several other tolerances and dimensional specifications. The specified region is filled with copper, with cutouts around any footprints, vias, or traces that are already there - unless the footprint/trace/via is supposed to be the same net as the fill, in which case the fill automatically connects.

   Press the "Add a filled zone" button on the right toolbar or press "CTRL+SHIFT+Z" and click the starting point of the plane. In the dialog, select the F.Cu layer and GND net. Changing the thermal relief and spoke to 0.75mm is recommended to improve the solder-ability of power net connections. After clicking "OK", draw the rest of the rectangle around the board, double clicking or clicking on the first point to finish the plane. Press "B" to autofill zones. This is seen in Figure 56. Do the same for the 5V net on the B.Cu layer.

   Lastly you will want a miniature power plane in the F.Cu layer near the amplifiers to connect them to the 5V power plane as they are on the top layer and generally a larger power connection is better. To make sure the plane fills with priority before the GND plane drawn before, make sure that the zone priority level field is a higher number. Once you have added this plane, add multiple vias from the plane down to the larger B.Cu layer.

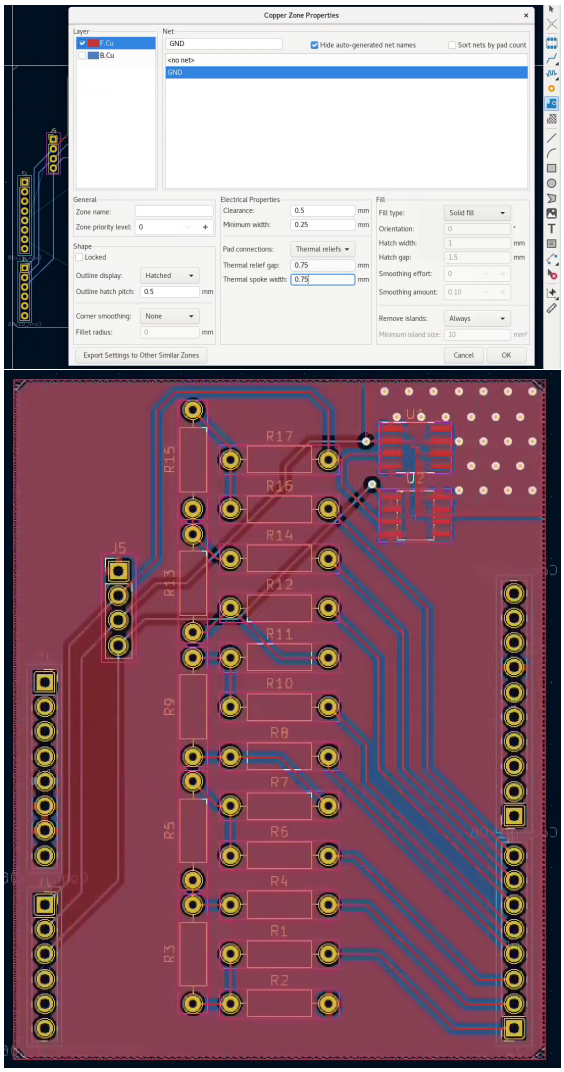

Figure 56: Adding the GND and power planes to the PCB.

9. Add a label silkscreen. Attach your name to the PCB so that it can be found in class. Use the text tool for this. See Figure 57. You should also label the outputs on the 4 pin header to easily connect to them layer. You can see in this figure that I have corrected the 5 position header and how minute of a difference it is from the previous 4 position header.

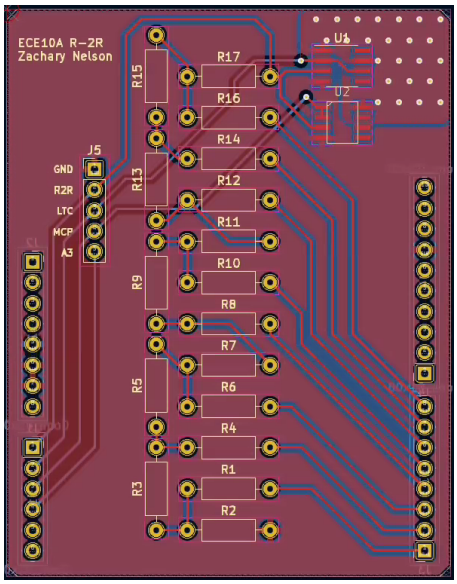

Figure 57: Adding labels using the Silkscreen label. Your finished board should look something like this.

10. When you think you are done, run a design rule check. This is accessible under the top menu, "Inspect - Design Rules Checker". Running the check should turn up no errors. If it does, you need to fix the errors in order for your PCB to be complete, for sending out the design files for fabrication. Figure 58 shows the DRC dialog, with a zero-error result returned.

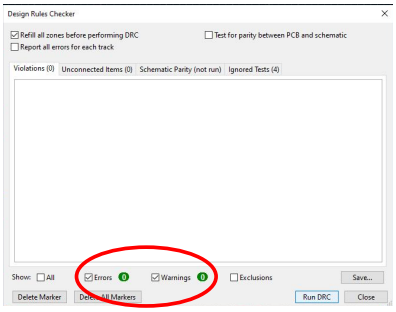

Figure 58: A clean design rule check (from a different PCB design).

11. After your design has passed DRC and your TA has checked it, send your design to a PCB manufacturer, so that we can assemble and test the PCB in a few weeks. Only one PCB design needs to be manufactured for each student pair - up to you whose design you go with.
    The standard interchange file format for giving a PCB design to a manufacturer is Gerber files. One Gerber file is used for each manufacturing layer, so you will end up with a number of files. To generate these, click the Plot button in the top toolbar on the left. Click the generate drill files first, then generate drill file, now close the drill file dialog. Make sure that you are plotting the front and back copper layers, silkscreens, paste, soldermask, and edge cuts layers at least, then click plot. You can see this in Figure 59. These files can be submitted to many manufacturers - some examples are Osh Park, Seeed Studio, or Bay Area Circuits.
    Osh Park is by far the easiest to use, and doesn't actually require that you export Gerbers. You can simply upload a KiCad PCB file, and Osh will process it for you. Furthermore, Osh will show you renderings of your PCB before you submit your order - good for double-checking your design. Their basic service ("Prototype") charges $5 per square inch of board, so making your board a little smaller can cut down on costs. These PCBs are usually finished in around two weeks (you can see all of these specs and turn times by exploring Osh's website), allowing plenty of time before we'll ask you to use them in lab (As of Fall 2023, the week of November 27).

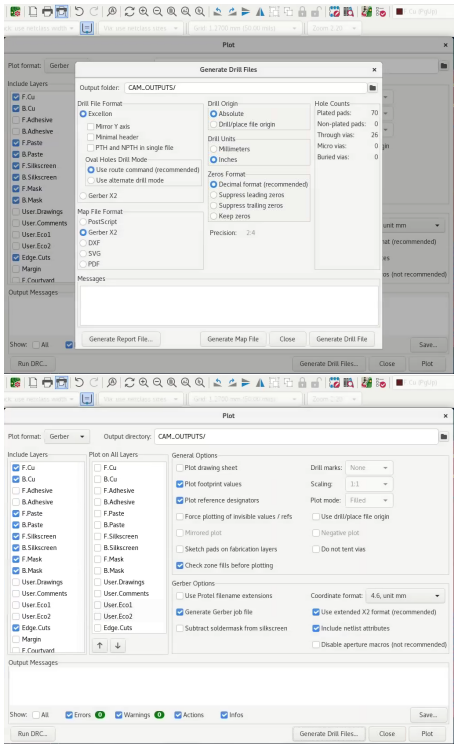

Figure 59: Gerber Generation