



# MOTOR DRIVEN DESK CLOCK

## SP6 Assembly Notes

Instructions for building a 3D printed desk clock  
using a custom stepper motor drive circuit

Steve Peterson  
30-Aug-2021

## Contents

Tables.....	2
Figures .....	2
Revision History.....	3
Description.....	4
Components.....	4
Important Note on Stepper Motors .....	5
Cut Metal Parts.....	7
Tools Required:.....	7
Motor Control Overview.....	8
Drive Circuit.....	8
Assembling the Motor Controller.....	11
Programming the Arduino Nano.....	15
Printing the Parts.....	16
Building the Clock.....	21
Component Pre-Assembly .....	21
Checking Component Fit.....	24
Adding the Gears.....	25
Calibration.....	31
Appendix A: Motor Control Experiments .....	32
Appendix B: Arduino Nano Algorithm .....	34
Appendix C: Some of my Other Clock Designs.....	37

## Tables

Table 1: Component list and print times: .....	17
--	----

## Figures

Figure 1: 30 $\Omega$ Stepper motor from StepperOnline .....	5
Figure 2: My favorite motor with a connector and a spline .....	6
Figure 3: 35 $\Omega$ Adafruit motor is not yet working.....	6
Figure 4: Cut metal parts list.....	7
Figure 5: Motor controller schematic.....	8
Figure 6: Stepper motor current .....	9
Figure 7: Initial test motor waveforms .....	10
Figure 8: Improved motor currents.....	10
Figure 9: Ideal stepper motor current .....	11

Figure 10: Motor controller parts .....	12
Figure 11: Arduino Nano with header posts installed .....	12
Figure 12: Motor controller with components added .....	13
Figure 13: Completed motor controller ready to add to Arduino Nano .....	14
Figure 14: Completed motor control module .....	15
Figure 15: Gear slicer detail with four perimeters .....	18
Figure 16: Front frame layer changes.....	18
Figure 17: Optional back frame layer changes.....	19
Figure 18: Color highlights on hands .....	19
Figure 19: Gear reference diagram .....	20
Figure 20: Gear 2 shaft assembly .....	21
Figure 21: Gear 5 shaft assembly .....	22
Figure 22: Segmented back frame assembly .....	23
Figure 23: Segmented front frame assembly.....	23
Figure 24: Stepper motor gear position .....	24
Figure 25: Add gear 2 assembly .....	25
Figure 26: Add gear 3.....	25
Figure 27: Add gear 4.....	26
Figure 28: Add gear 5 assembly .....	26
Figure 29: Knob added to gear 5 behind the clock.....	27
Figure 30: Add gear 6.....	27
Figure 31: Add gear 7.....	28
Figure 32: Add gear 8.....	28
Figure 33: Add the front frame .....	29
Figure 34: Motor and electronics added to base .....	30
Figure 35: Fully assembled clock.....	30
Figure 36: AC synchronous motor .....	32
Figure 37: Microwave oven synchronous motors.....	32
Figure 38: Small geared stepper motors .....	33
Figure 39: Grasshopper clock and wood clock rendering.....	37
Figure 40 Wooden gear experiments .....	38
Figure 41 Desk clocks.....	38
Figure 42 Original Thingiverse design.....	39
Figure 43: Large pendulum clock .....	40
Figure 44: Easy build clock with 32 day runtime.....	41
Figure 45: Electric pendulum drive .....	42

## Revision History

30-Aug-21      Original version

## Description

This is the assembly guide for a 3D printed desk clock powered by an electric motor. It is my first clock design that never needs winding. Just provide a USB power source and the clock will run forever. The design used the following criteria:

- 1) The clock should look good.
- 2) It must be accurate.
- 3) It should be quiet.
- 4) A sweep second hand is good.
- 5) The design should not be dangerous. No high voltages. The motor should have a low enough torque that it stalls if anything is stuck in the gears. etc.

Large exposed gears help satisfy the first criteria to look good. If you just want a box with hands and a dial, then buy a clock from Walmart. The remaining criteria make this a functional clock that can sit anywhere. An Arduino Nano controls the timing to a custom stepper motor controller to make the clock extremely quiet.

I designed the clock using a Prusa MK3S, but any printer with at a 170x160mm print area will work.

## Components

This clock uses a few critical components, in addition to the printed parts. The most important are the Arduino Nano, NEMA17 stepper motor, and a custom motor control circuit.

Here are the non-printed parts required to build the clock:

Component	Notes
Arduino Nano without pre-soldered headers	The complete part description is "USB CH340G Nano V3.0 16M ATmega328P Micro-Controller Board for Arduino".
Mini USB cable	For powering and programming the Arduino Nano which uses the old style mini USB instead of the more common micro USB cable.
USB power source	To power the clock while running. Current is less than 100mA.
NEMA17 stepper motor	It is important to get a motor with 30-35Ω coils. See the next section for more details.
custom motor controller	Available for a nominal fee at the link in the description of this clock on MyMiniFactory.
4X 6x3/4" flat head wood screws (metric equivalent is M3.5x20mm)	Frame is assembled using small flat head wood screws. Add three additional screws if printing the segmented frame on a small printer.
18" X 1/16" music wire (metric equivalent is 0.5m X 1.5mm)	Arbors are made using hardened music wire rod cut into short segments (described later). Add an additional 18" (0.5m) in length if printing the segmented frame on a small printer.
1X small spring	Use a spring from a ball point click pen for the friction clutch to set the time.
4X M3x10mm screws	Socket heads are best, but most small diameter head styles will work.
4X 1" diameter felt pads	Optional pads for under the base of the clock.

## Important Note on Stepper Motors

The stepper motor must have high resistance coils of 30-35Ω to work with the low noise stepper motor controller. They are not the most common stepper motor style, but do appear to be readily available. Motors with high current 1-3Ω coils will not work. If they don't mention the resistance, but have other descriptions like 1.6A, then they are likely low resistance coils that will not work.

It seems like 30-35Ω motors were designed to be driven with simple transistor switches like a ULN2003 and a 12V supply. ULN2003 drivers could have been made to work in the clock, but the jerking action of each step would make the clock really noisy. The custom designed stepper motor controller smooths out the motion with micro-stepping. The current levels are low enough to run from the 5V USB power source that powers the Arduino. There is still plenty of power to run the clock.

A good search description for the stepper motors is "NEMA17 stepper motor" combined with "12V", or "0.4A" or "0.35A". The maximum body length is 43mm, although the most common motors will be around 34mm long. Single shaft is best. Dual shaft might not work since the second shaft would interfere with the front frame. The controller uses four wires. It can work with six wire motors by ignoring the center tap connections.

Here are a few motors that have been tested to work:



Roll over image to zoom in

### STEPPERONLINE 17HS13-0404S1 Stepper Motor for 3D Printer DIY CNC Robot, -10-50 Degree C, 0.4 Amp, Black

Visit the [STEPPERONLINE Store](#)

★★★★★ 95 ratings | 15 answered questions

Price: **\$10.99** & **FREE Returns**

Available at a lower price from [other sellers](#) that may not offer free Prime shipping.

- Most Popular Nema 17 Stepper Motor
- Brand STEPPERONLINE
- Holding Torque: 26Ncm(36.8oz.in)
- Nema 17 size 42x42x34mm
- 0.4A with 12V dc voltage
- 1m Cable w/ Connector

#### Specifications for this item

Brand Name	STEPPERONLINE
Cable Length	39.37 inches
Coil Resistance	30 ohm
Color	black
Connector Type	1" Pitch Connector
Current Rating	0.4 amps
Ean	0606682999450 , 0701056302451
Height	1.65 inches

▼ See more

Figure 1: 30Ω Stepper motor from StepperOnline

This motor with a built-in spline is a great option. The first gear in the clock is provided in two styles, one that slides directly onto the spline, and one that fits a standard 5mm shaft.



Minebea 2-Phase NEMA 17 Stepper Motor  
10.2V 1.8 Degree 4-Wire 6-Pin 34 Ohms  
0.3A

★★★★★ Be the first to [write a review](#).

Condition: **New**

Quantity:  120 available

Price: **US \$7.99**

[Buy It Now](#)

[Add to cart](#)

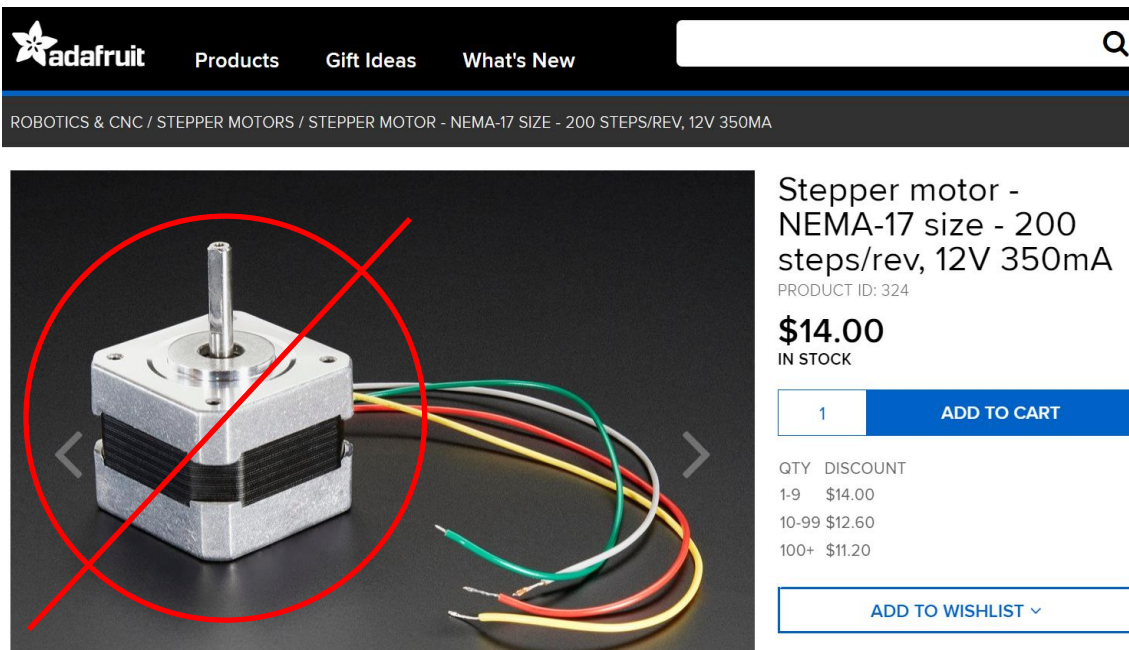
[Add to Watchlist](#)

**Same day shipping**    30-day returns    Ships from United States

Shipping: \$5.49 Standard Shipping | [See details](#)  
Located in: Springfield, Tennessee, United States

Figure 2: My favorite motor with a connector and a spline

The 35 ohm motor from Adafruit spec looks good, but I was not able to get it to work smoothly. Maybe it needs different tuning in the algorithm. Hold off on using this motor for now.



adafruit    Products    Gift Ideas    What's New   

ROBOTICS & CNC / STEPPER MOTORS / STEPPER MOTOR - NEMA-17 SIZE - 200 STEPS/REV, 12V 350mA

Stepper motor -  
NEMA-17 size - 200  
steps/rev, 12V 350mA  
PRODUCT ID: 324

**\$14.00**  
IN STOCK

[ADD TO CART](#)

QTY    DISCOUNT  
1-9    \$14.00  
10-99    \$12.60  
100+    \$11.20

[ADD TO WISHLIST](#)

Figure 3: 35Ω Adafruit motor is not yet working



## Cut Metal Parts

The music wire should be cut to the following lengths and burrs removed from the ends. The design works equally well with 1/16" or 1.5mm music wire since they are very close to the same size. Music wire comes in a hardened state which is great for the clock, but difficult to cut. A Dremel cut-off disk or cutters with hardened jaws will work well. Cheap wire cutters might not be tough enough.

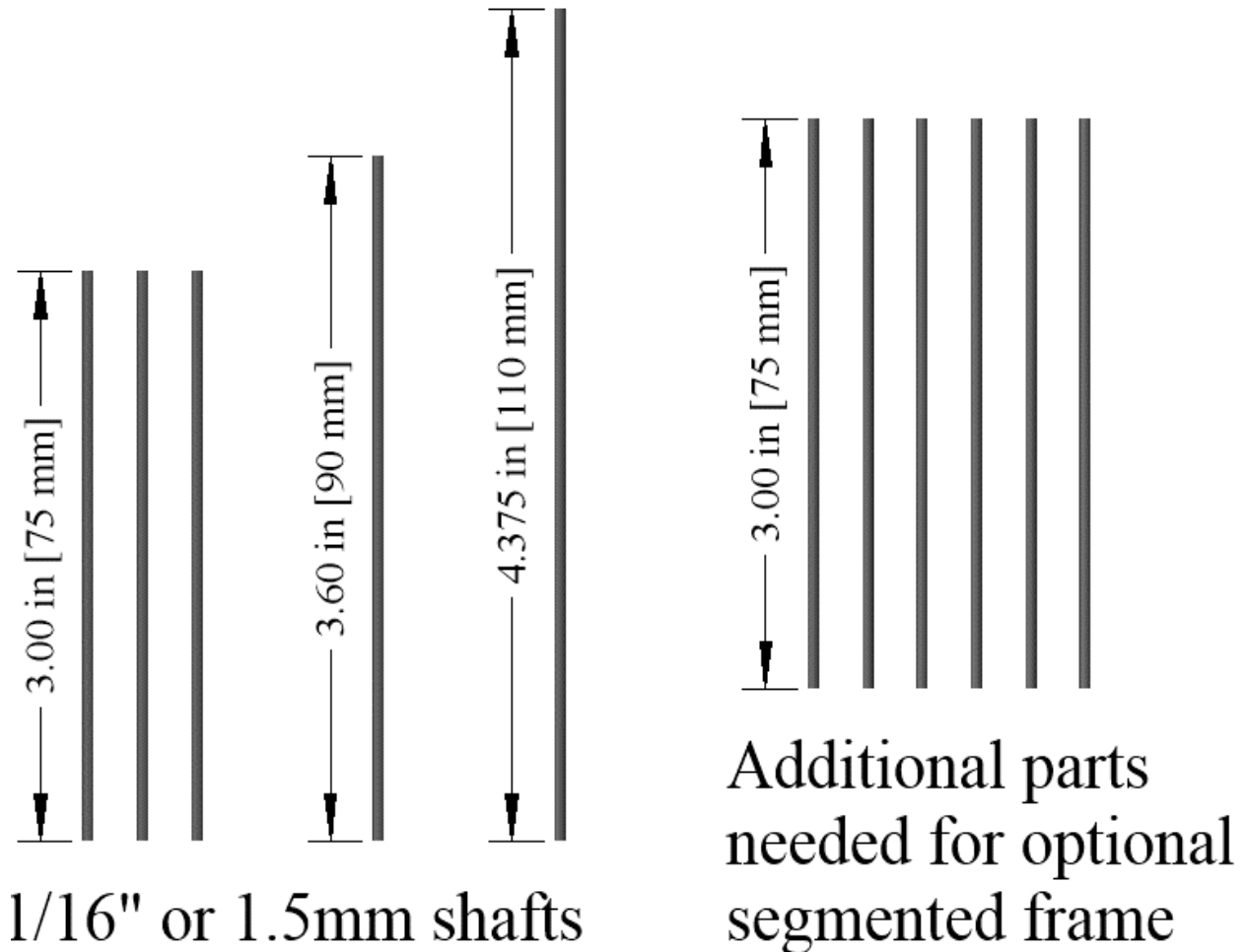


Figure 4: Cut metal parts list

## Tools Required:

A few simple tools are required for building the clock.

- 1) Phillips screwdriver
- 2) Hex wrench for M3x10mm screws
- 3) Hacksaw or Dremel cut-off disks for cutting music wire
- 4) Soldering iron for assembling stepper motor driver
- 5) Ohm meter for verifying resistor values is nice
- 6) Fine tooth hand files or sandpaper may be needed to clean up some of the printed parts
- 7) 1.5mm or 1/16" drill bit for cleaning up printed holes
- 8) Small hammer

## Motor Control Overview

I tried many stepper motor control methods before designing my own custom circuit. This was the best way I could find to make the clock run quietly. All the conventional stepper motor drivers were jumpy and the gears would rattle. The custom circuit allows each motor coil to be driven with 32 current levels and 250 time steps per quadrant. The result is very close to a true sine wave with smooth motion. I encourage builders to improve the design or find a ready-made solution that works just as well and share it with others.

An Arduino Nano drives the motor directly through series resistors to keep the current within the safe operating range. The 30-35 $\Omega$  motor is the only type motor that works with the low current driver. Low resistance motors would need higher current than the Arduino can provide.

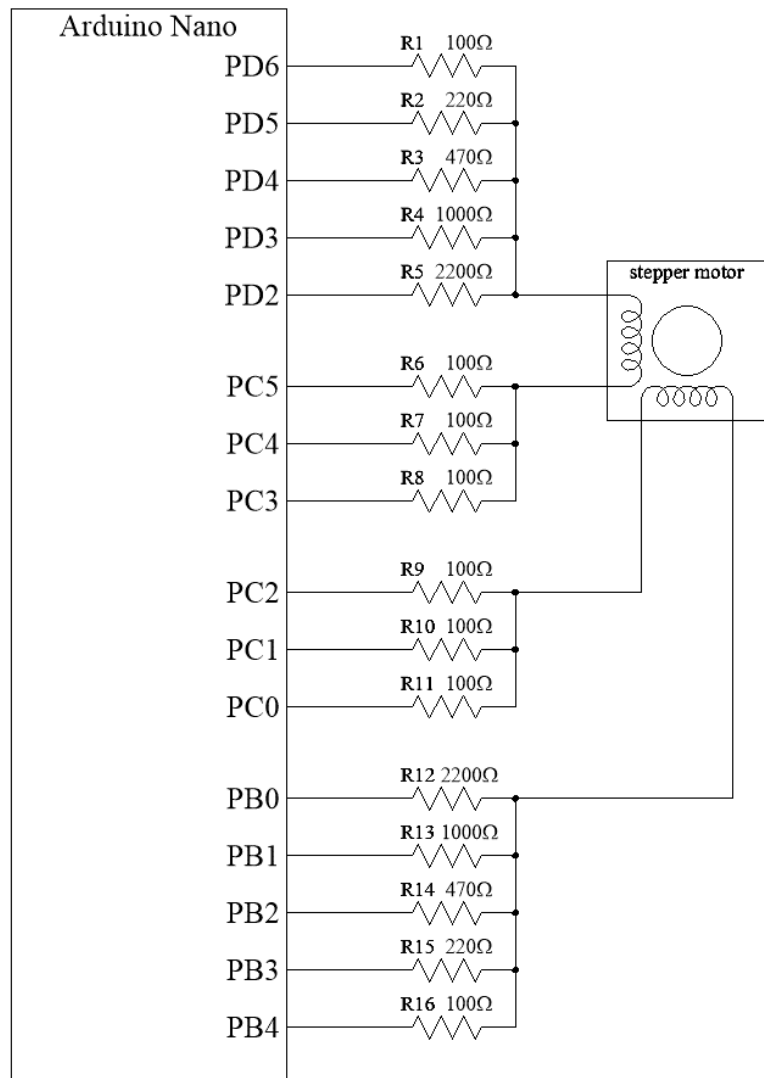


Figure 5: Motor controller schematic

## Drive Circuit

The drive circuit is shown above. It uses sixteen of the Arduino output pins to directly drive the motor. Parallel pins limit the current through any individual Arduino output driver. The banks of 5 pins are binary weighted to provide 32 levels of proportional control. The banks of 3 pins are designed to share the current.

The stepper motor current can range from 2.4mA when a single pin is driving to a maximum of 33mA when all five pins are driving. The maximum current through a single pin was measured at 22mA when only PB4 or PD6 is driving. This is still below the maximum limit of 40mA per output pin of the Arduino Nano. This circuit has been tested for several months so far without failures.



There are several advantages to running such low currents through the stepper motors. Torque is very low, but still has more than enough power to run the clock. You can stick your fingers into the gears and the motor will stall before chopping off a finger. This satisfies the criteria that the clock should be safe even with the exposed gears. The motors are rated at 350-400mA, but running on less than 33mA, so the motors have less than 10% of their potential torque. This keeps them cool and allows them to run from a simple USB power source.

The low motor currents allow the clock to be powered and programmed with a USB cable. The total power consumed by the clock is around 65mA consisting of 19mA for the Arduino Nano plus an average of 23mA for each of the two stepper motor coils. This is well below the 100mA limit of any USB port. After programming, the clock runs on a small USB charging adapter.

The following currents were measured for each of the 32 different current settings. The resistor sizes have been selected to make the currents reasonably linear. The algorithm can compensate for any slight non-linearities.

Each motor coil can receive any of 32 current levels. Both coils operating together provides 64 unique motor positions between steps. This would be considered 64 level micro-stepping.

The motor has 200 full steps per revolution, so the micro-stepping driver is capable of positioning to 12800 positions per revolution. The algorithm increases the time resolution to 50000 steps per motor revolution.

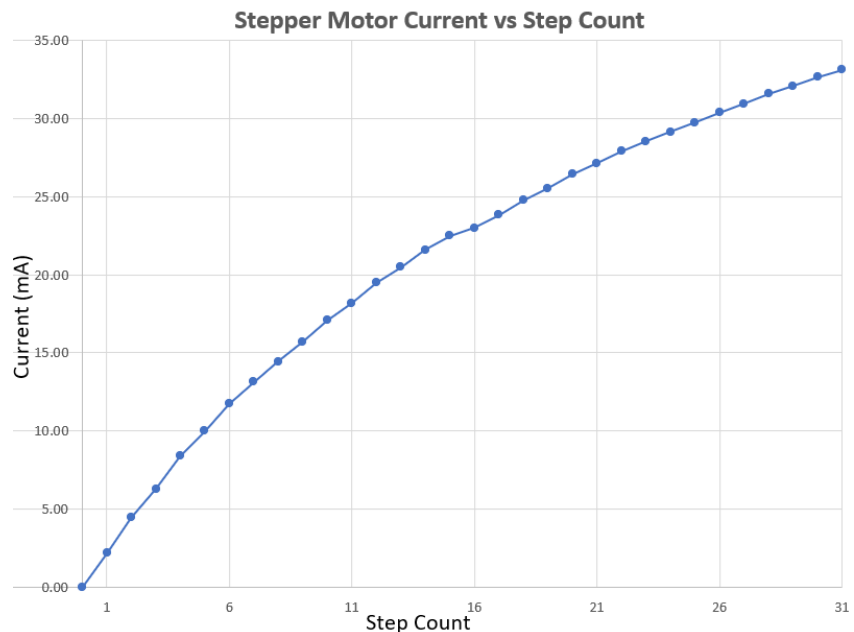
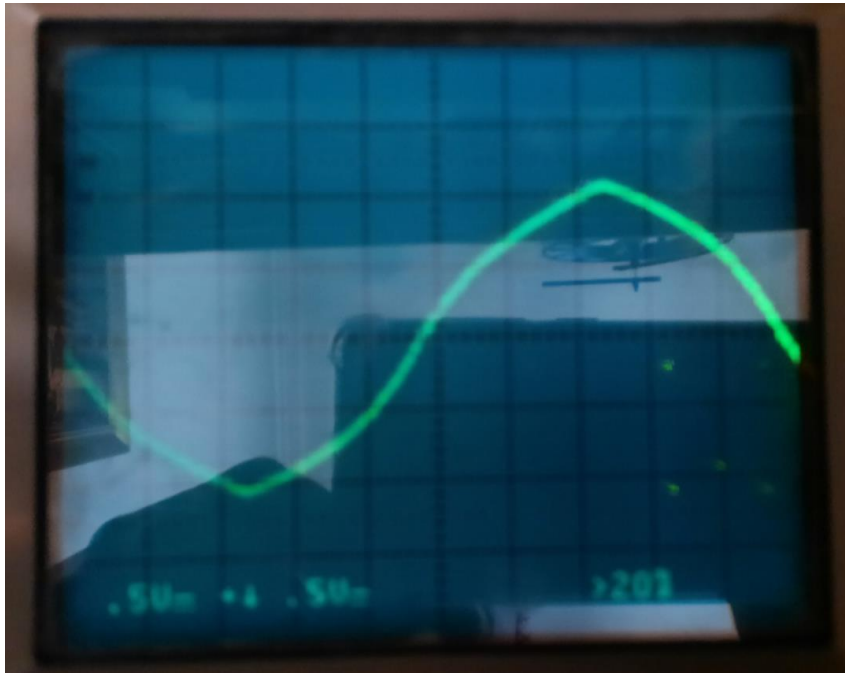


Figure 6: Stepper motor current

The first algorithm used to drive the clock had a simple delay routine to step through the 32 different current levels. The algorithm repeats for each of the four quadrants of a sine wave. The second motor coil is shifted 90 degrees to produce a cosine wave. During each cycle, one bank of 5 pins counts up while the other bank counts down. The cycles alternate between sourcing and sinking current. This creates a close approximation of a sine wave current through the stepper motor coil.

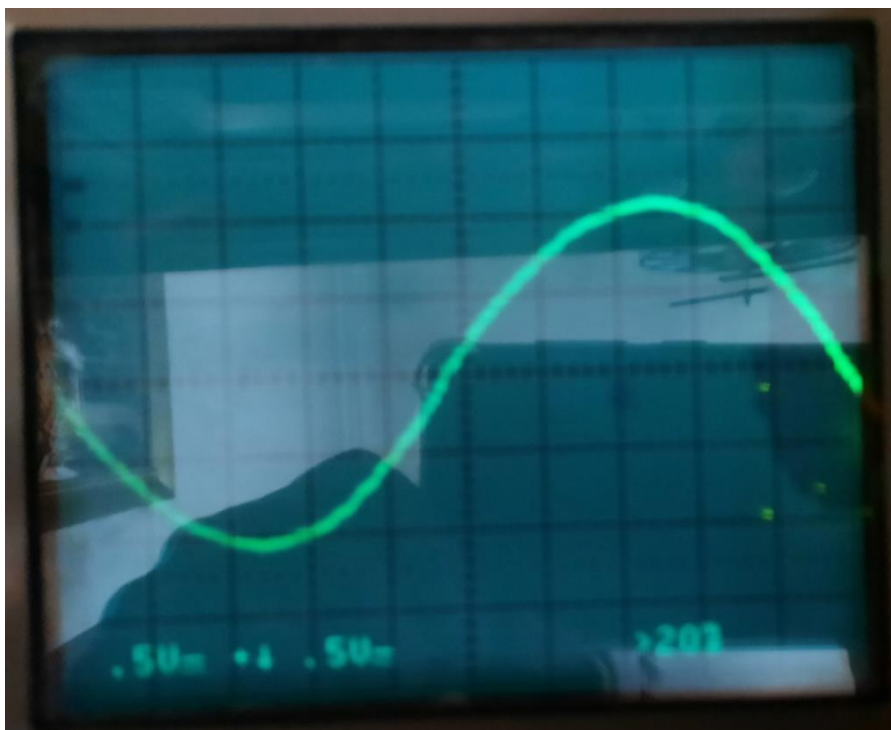
The resulting current through the coils is shown below. The clock runs fairly smooth and is reasonably quiet, but there is a slight rumbling noise that I attribute to the waveforms not being perfectly



sinusoidal. It appears that the sharp current peaks at the top and bottoms of the wave are causing the motor to jump slightly during portions of the cycle. A better algorithm was created to produce a more sinusoidal waveform.

Figure 7: Initial test motor waveforms

The latest algorithm uses variable time delays between steps to match a sinusoidal curve as closely as possible. Each quadrant of the waveform has been modified to use 250 discrete time steps. A 250 entry



lookup table selects the best current level for each moment of time. The resulting waveform is a very close approximation of a sine wave and the clock runs much quieter.

Figure 8: Improved motor currents

The description of ideal microstepping current levels was found at [Microstepping Tutorial - Technical Article - Zaber](#). It describes a current profile where the current transitions in a sinusoidal fashion between steps. The description assumes a motor controller that is able to modulate the current as needed. However, my driver has fixed current levels determined by the resistors, so the algorithm was modified to change the time between current steps.

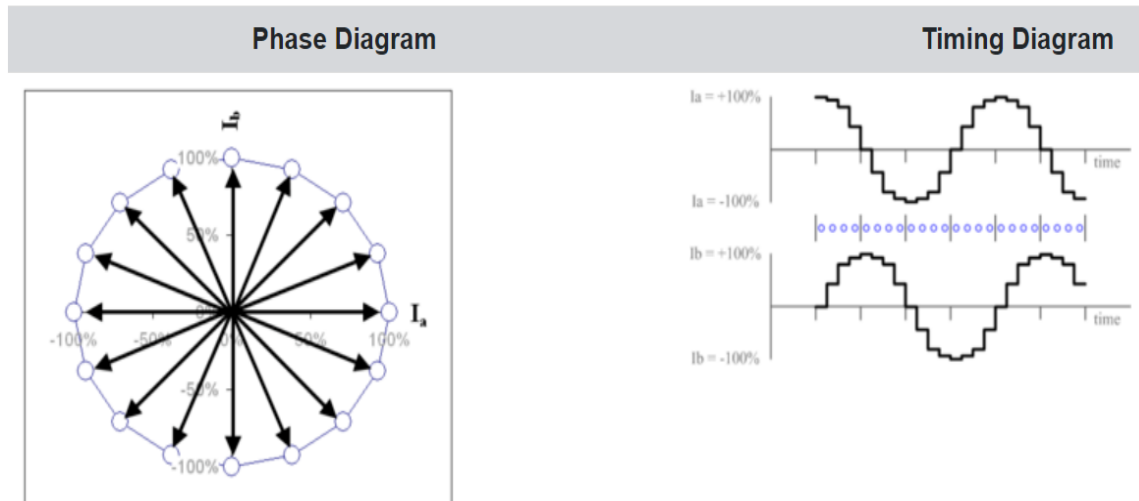


Figure 9: Ideal stepper motor current

### Assembling the Motor Controller

The motor controller uses a small circuit board that mounts onto the Arduino Nano. Extension headers make assembly easier and allows the Arduino to be swapped if needed. The motor controller kit is available for purchase from the link in the clock description on MyMiniFactory. It includes the circuit board, resistors, and extension headers. You will need to solder the board.

The optional 4 pin terminal block is too thick for standard letter postage, so it can be provided, but the shipping cost is very high for such a small part. I am not a huge distributor with negotiated shipping rates, just a normal person using standard postal rates.

I prefer to not be in the business of buying and re-selling Arduino Nanos or stepper motors, so you will have to source them separately. The Arduino Nano is easily found online at reasonable prices in single quantities. Make sure to get the Arduino Nano, not the smaller Arduino Mini or other variants. The controller board was designed for the Arduino Nano footprint.

The stepper motor is available on eBay, Amazon, AliExpress or other standard places. Make sure to select a 4 wire NEMA17 motor with 30-35Ω coils. A body length around 34mm is ideal. They are often listed as 12V or 0.4A motors. A connector plug on the motor is nice, but not critical.

The motor controller kit has 16 resistors, two 15 pin extension headers, and a small circuit board. Both sides of the board are shown, but obviously only one board is included. The optional terminal block allows the motor to be removed easily, but is not absolutely required. The motor wires can be soldered directly to the board.

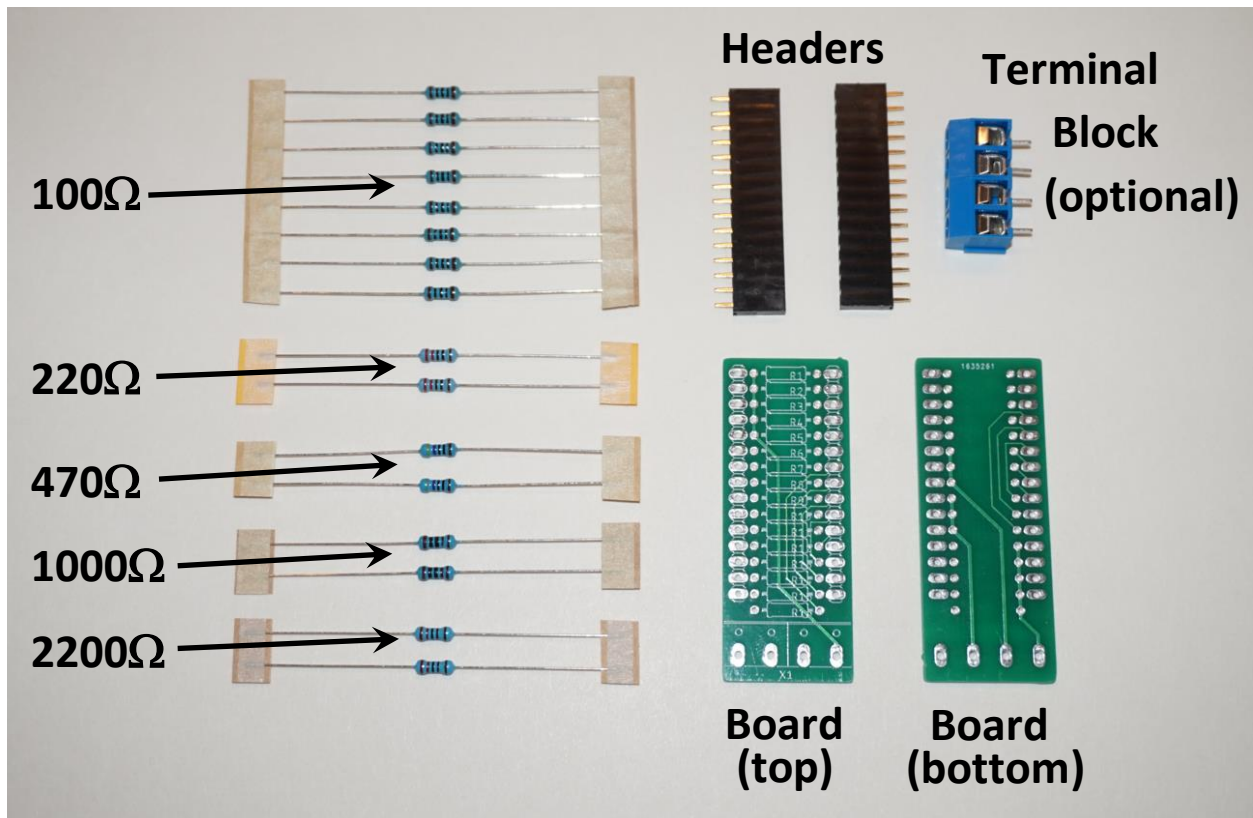


Figure 10: Motor controller parts

Step 1) Solder the 15 pin posts to the TOP side of the Arduino Nano. The pins need to be facing up. The six pin header is not needed since programming is done through the USB port.

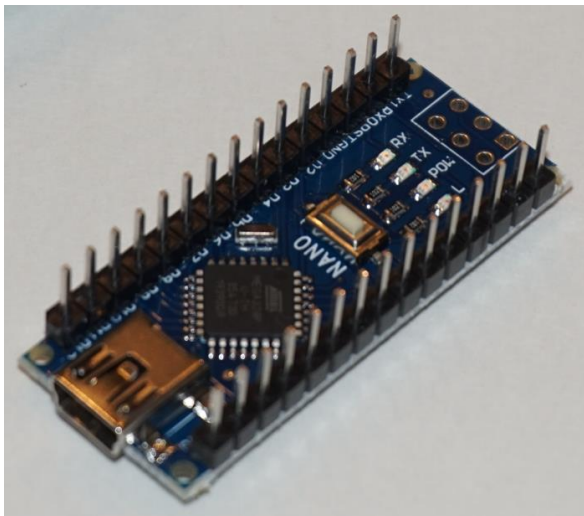


Figure 11: Arduino Nano with header posts installed

Step 2) Solder the resistors into the controller board. The order of the resistors is the same as shown on the schematic, 100Ω, 220Ω, 470Ω, 1000Ω, 2200Ω, 100Ω, 100Ω, 100Ω, 100Ω, 100Ω, 100Ω, 2200Ω, 1000Ω, 470Ω, 220Ω, and 100Ω. The color coding on the 1% tolerance resistors in the kit is hard to read. Use an ohm meter to verify the values if needed.

Step 3) Solder extension headers to the bottom of the controller board. Tack the corner pins first and make sure the header is straight. Go slow to prevent melting the plastic part.

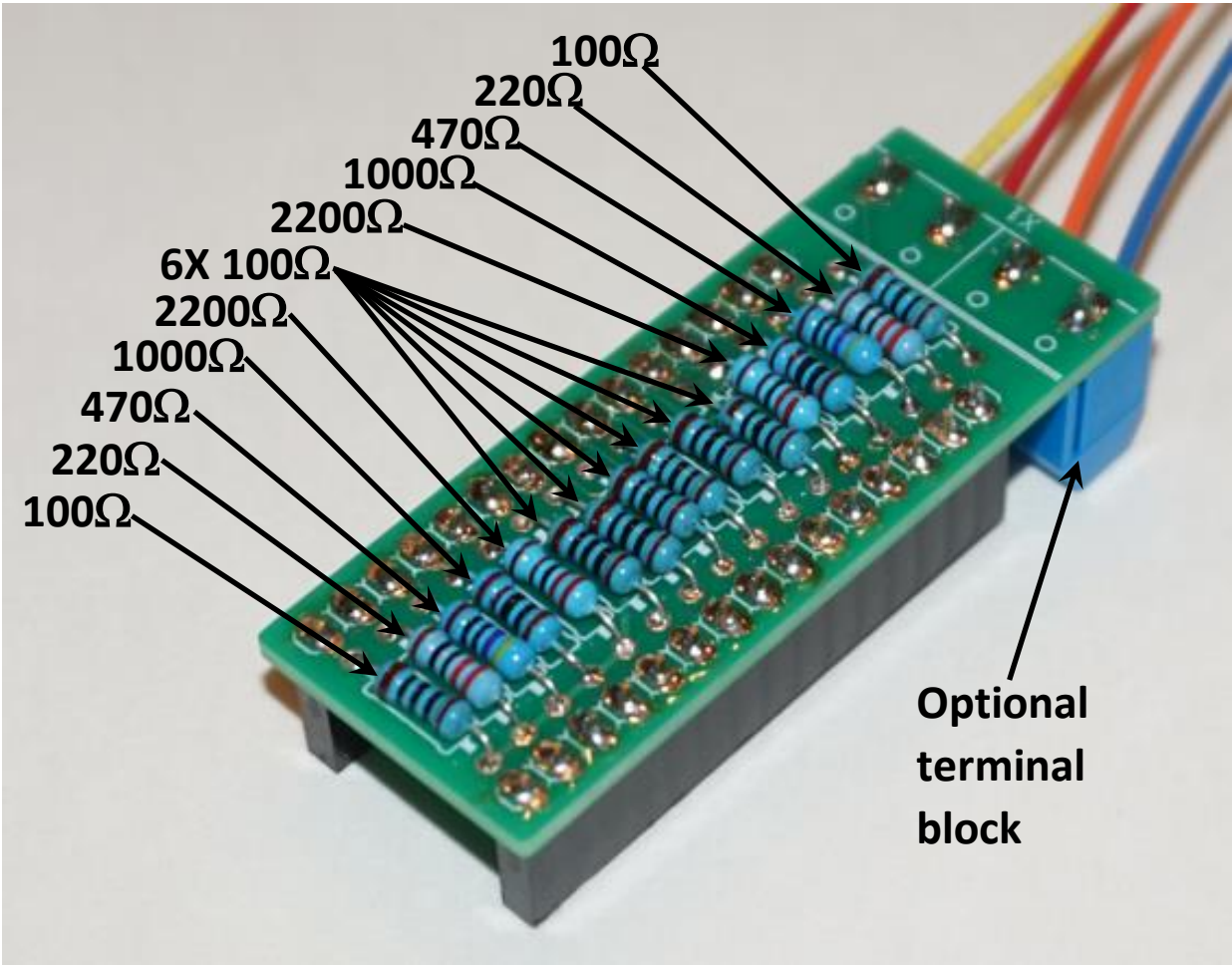
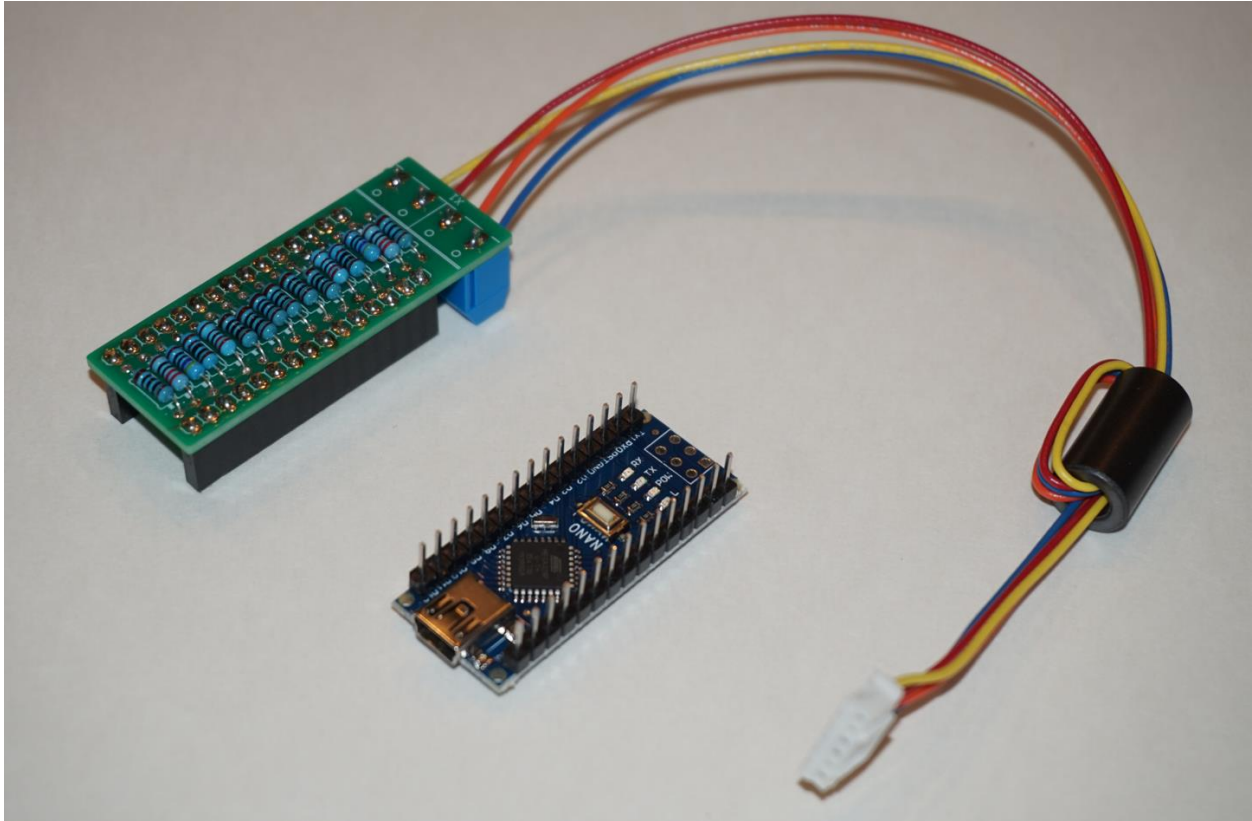


Figure 12: Motor controller with components added





*Figure 13: Completed motor controller ready to add to Arduino Nano*

Step 4) Connect the stepper motor to the controller board. The 4 pin terminal block is nice, but not required. It is OK to solder the motor wires directly to the motor controller, especially if there is a connector at the motor. Connect the motor wires with one coil on the left two pins and the other coil on the right two pins. Use an ohm meter to check the coils. It doesn't matter how the two wires in each motor coil are connected. Swapping them will make the motor run backwards, but the algorithm can easily reverse the direction. The most important connection is to have the wires for one coil on the left two pins and the other coil on the right two pins.



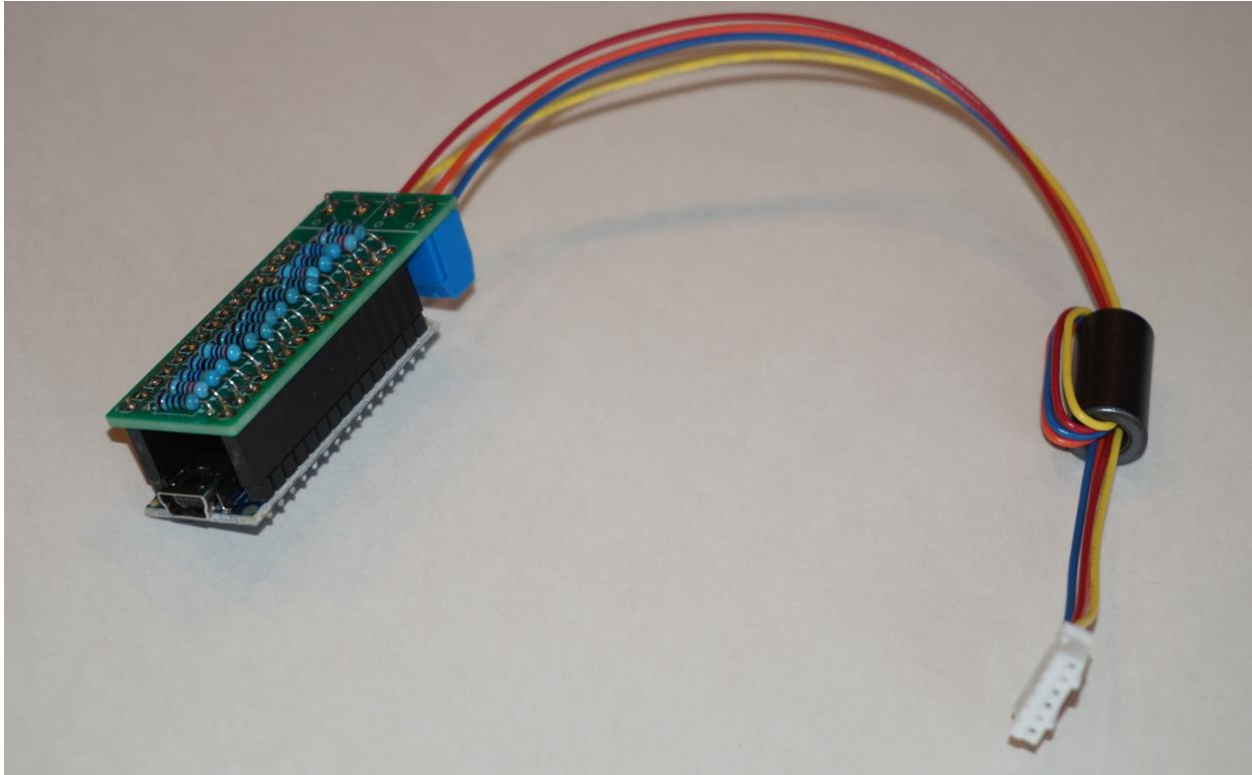


Figure 14: Completed motor control module

Step 5) Plug the controller onto the Arduino Nano and connect the stepper motor. The mini USB plug should be opposite from the motor wires. The motor will not be spinning yet, since the Arduino still needs to be programmed. Some builders caution against connecting or disconnecting a stepper motor while the controller is powered, so I always unplug power to the Arduino before connecting the motor.

## Programming the Arduino Nano

Nearly every pin of the Arduino Nano is used to drive the stepper motors. The code assumes that the Arduino will only be controlling the stepper motor. There is no multi-tasking, although it is conceivable that other functions could be added. There are a few unused pins that could be connected to circuits such as a real time clock module for greater timing accuracy. Enhancements like this may be available in the future.

Here are the steps to program the Arduino Nano:

Step 1) Download the Arduino IDE at <https://www.arduino.cc/en/software> selecting the option for Windows, Linux, or Mac.

Step 2) Plug the Arduino Nano to your computer using a mini USB cable.

Step 3) Open the IDE and configure for Arduino Nano

Step 3a) Tools → Board → Arduino Nano

Step 3b) Tools → Processor → ATmega328P (Old Bootloader)

Step 3c) Tools → Port → “Varies, usually COM3 or COM4 on my PC”

Step 3d) Tools → Programmer → USBasp

Step 4) Copy the algorithm from the appendix and paste it into the IDE window.

Step 5) Verify (compile) the code by clicking the “check mark” icon in the upper left corner of the IDE.

Step 6) Upload the code by clicking the “→” icon in the upper left corner of the IDE. The Arduino LEDs should blink for a second or two as the algorithm is uploaded, although they are somewhat hidden under the motor control circuit.

The stepper motor should start rotating 4 to 5 times per minute if it is connected and programmed. The first stage of gearing reduces the speed to once per minute. The motor control circuit is deliberately weak, so you can easily stall the motor by holding the output shaft. It should rotate smoothly with no noticeable steps.

Calibration of the exact rotational speed will take place after the clock is built.

## Printing the Parts

Print one of each clock part from the following table. Some parts are provided with multiple configurations, so select the one that works best for you.

I print everything using PLA with a 0.4mm nozzle, 0.2mm layer heights, 4 perimeters, 6 bottom layers, 7 top layers, 20% cubic infill, random seams, and 0.12mm elephants foot compensation. The default orientation is usually optimal with the largest surface already facing down. Supports are never needed.

A few parts have options depending on your printer size or other requirements. The base can be printed with the power plug at the back or the side if you want to place the clock on a narrow shelf. The base is symmetrical so it can be rotated for power on either the right or the left.

The frame is provided as a large piece or segmented into components that can be printed on smaller printers. The full one-piece frame needs a 205x160mm print area. It can be printed in sections on a printer with at least a 170x160mm print area.

The stepper motor gear has two different center hubs. One version fits directly on the 14 tooth spline that comes on some stepper motors. The other option fits a standard 5mm shaft using a M3x10mm set screw.

File Name	Color Suggestions	Print	Time	Filament	Notes
base_back_power	Tan	1	9h 20m	46.23	Base with power in the back
base_side_power	Tan	0	9h 22m	46.29	Optional base with power on the side
frame_back	Tan, Gold	1	5h 9m	21.01	Full back frame (190x150mm)
frame_back_split_bottom	Tan, Gold	0	3h 27m	13.04	Optional split back frame for smaller printers (150x130mm)
frame_back_split_top	Tan, Gold	0	2h 21m	9.70	
frame_front	Tan, Ivory, Black	1	6h 27m	30.76	Full front frame (205x160mm)
frame_front_split_left	Tan	0	0h 43m	2.70	Optional split front frame for smaller printers (160x160mm)
frame_front_split_right	Tan	0	0h 43m	2.71	
frame_front_split_top	Tan, Ivory, Black	0	5h 51m	27.67	
gear1_15_hub	Gold	1	1h 1m	2.75	Gear 1 with spline center hub
gear1_15_screw	Gold	0	1h 4m	2.78	Optional gear 1 with set screw
gear2_54_15	Gold	1	3h 25m	11.41	Needs to be tight on shaft
gear3_45_20	Gold	1	2h 54m	9.24	
gear4_50_25	Gold	1	3h 4m	10.59	
gear5_12	Gold	1	0h 36m	1.81	Needs to be tight on shaft
gear5_50	Gold	1	1h 30m	5.14	
gear5_insert	Gold	1	0h 21m	0.39	Needs to be tight on shaft
gear5_knob	Gold	1	0h 21m	1.47	
gear5_spacer	Gold	1	0h 4m	0.21	
gear5_washer	Gold	1	0h 1m	0.03	
gear6_48_15	Gold	1	2h 43m	7.84	
gear7_45_12	Gold	1	1h 32m	5.64	
gear8_48	Gold	1	1h 36m	5.05	
hand_hour	Ivory, Black	1	0h 10m	0.47	
hand_minute	Ivory, Black	1	0h 11m	0.51	
hand_second	Gold	1	0h 11m	0.49	
motor_mount	Tan	1	1h 57m	9.86	
<b>Total</b>		<b>20</b>	<b>42h 56m</b>	<b>170.79</b>	

Table 1: Component list and print times:

Most parts are optimized to print using 4 perimeters with a 0.4mm nozzle. The gear teeth and spoke widths will print almost completely solid using only perimeters. The total print time and filament usage is nearly the same as two perimeters, but the gears are much stronger. Four perimeters is also used on the frame to provide additional strength. I set up my slicer with 4 perimeters as the default and rarely change it. The following picture shows how the gears look in the slicer. Notice that there is no infill, only perimeters.

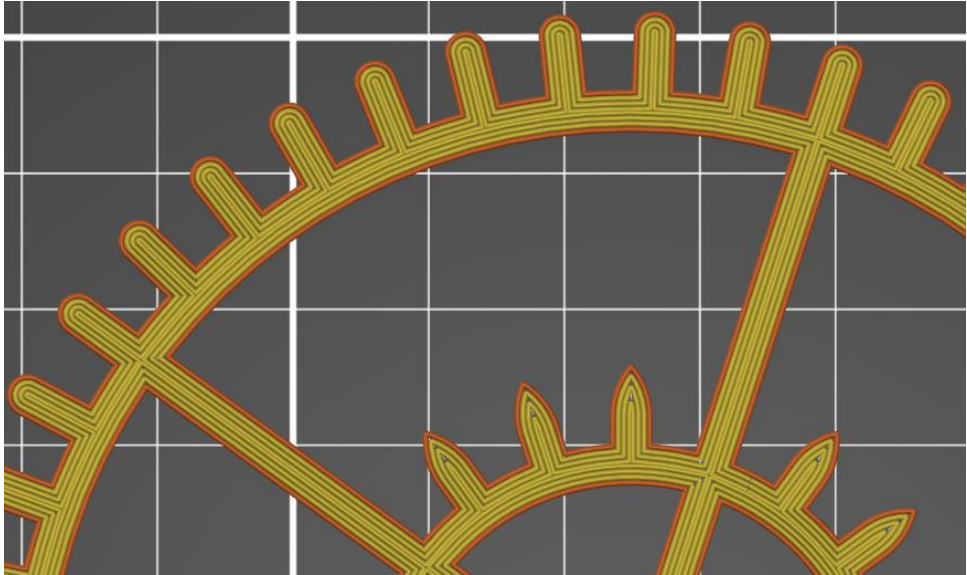


Figure 15: Gear slicer detail with four perimeters

The frame\_front needs a color change at 11.60mm to add highlights for the numbers. An additional color change at 10.40mm to change the dial to a light color is optional depending on the base color.

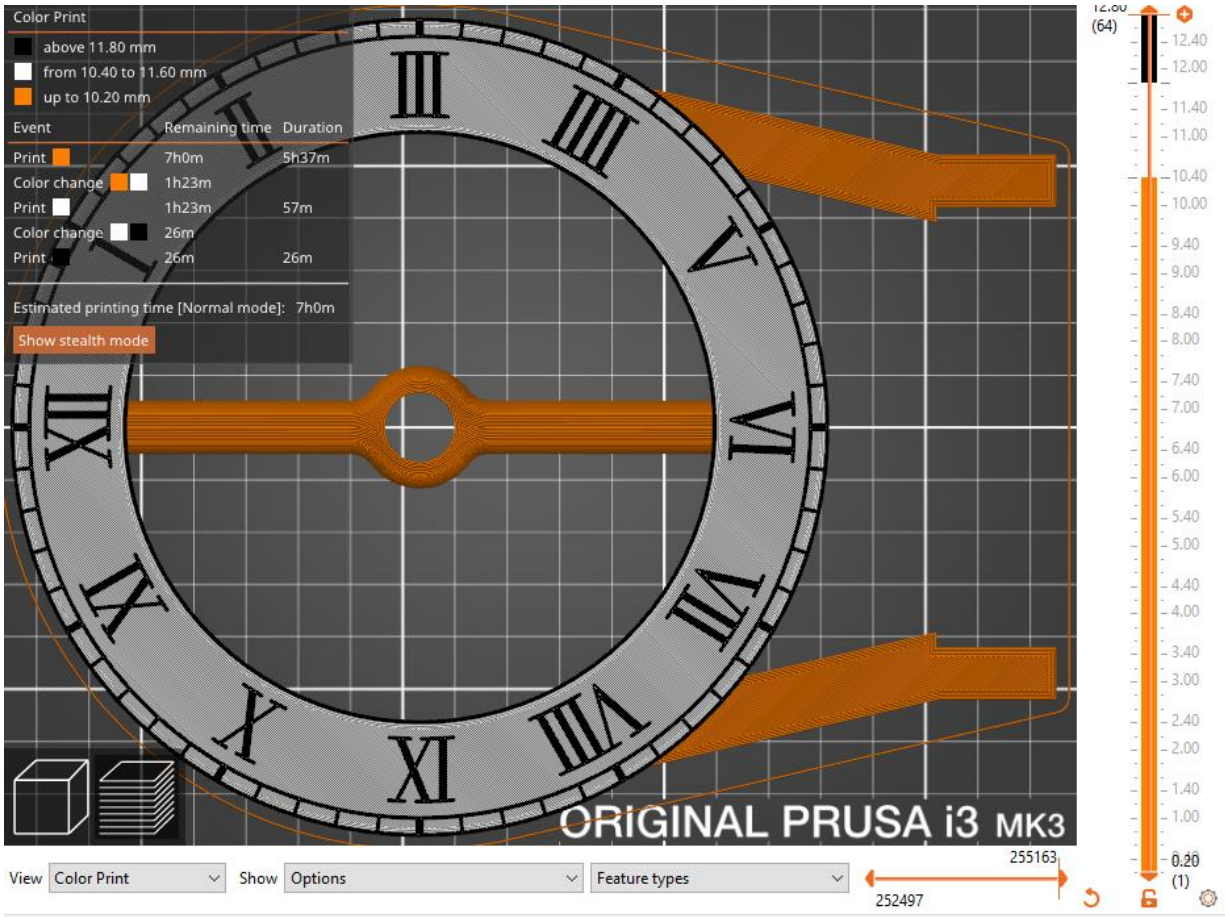


Figure 16: Front frame layer changes



The back frame has integrated columns to position the gears to the proper heights. The clock looks best with a color change at 10.40mm to match the gear color.

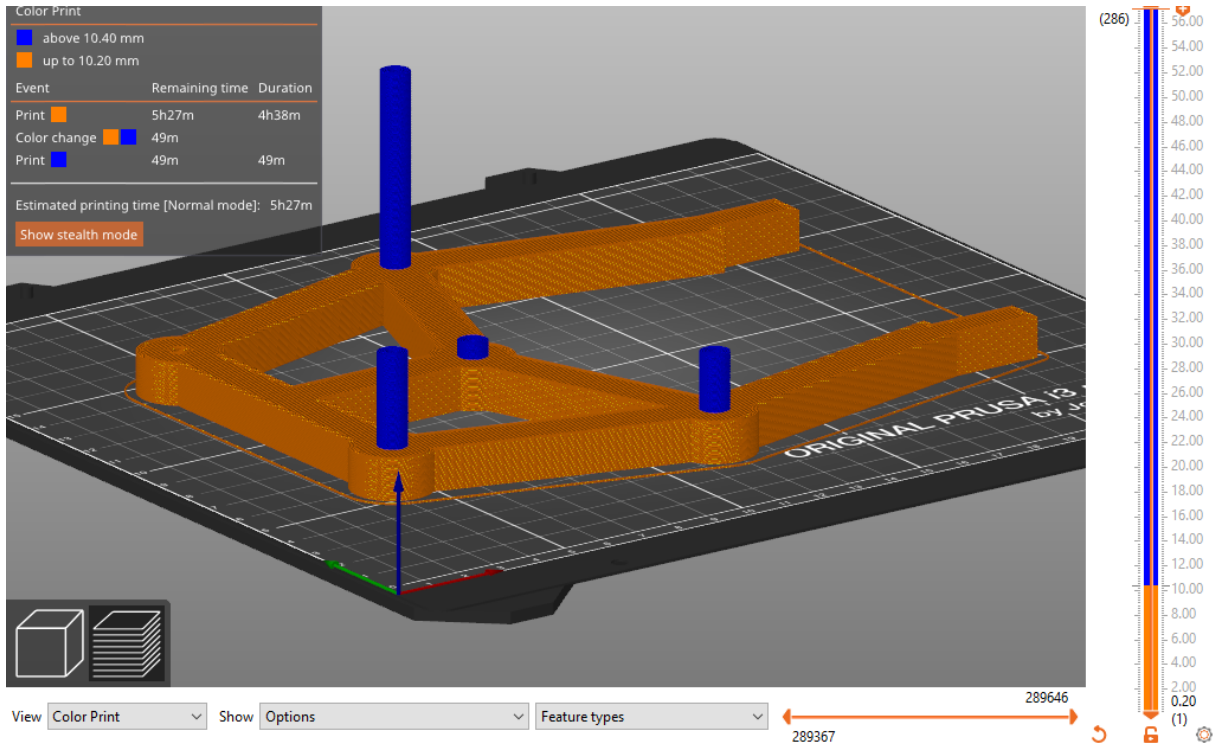


Figure 17: Optional back frame layer changes

The hour and minute hands can have a color change at 2.20mm to add some highlights.

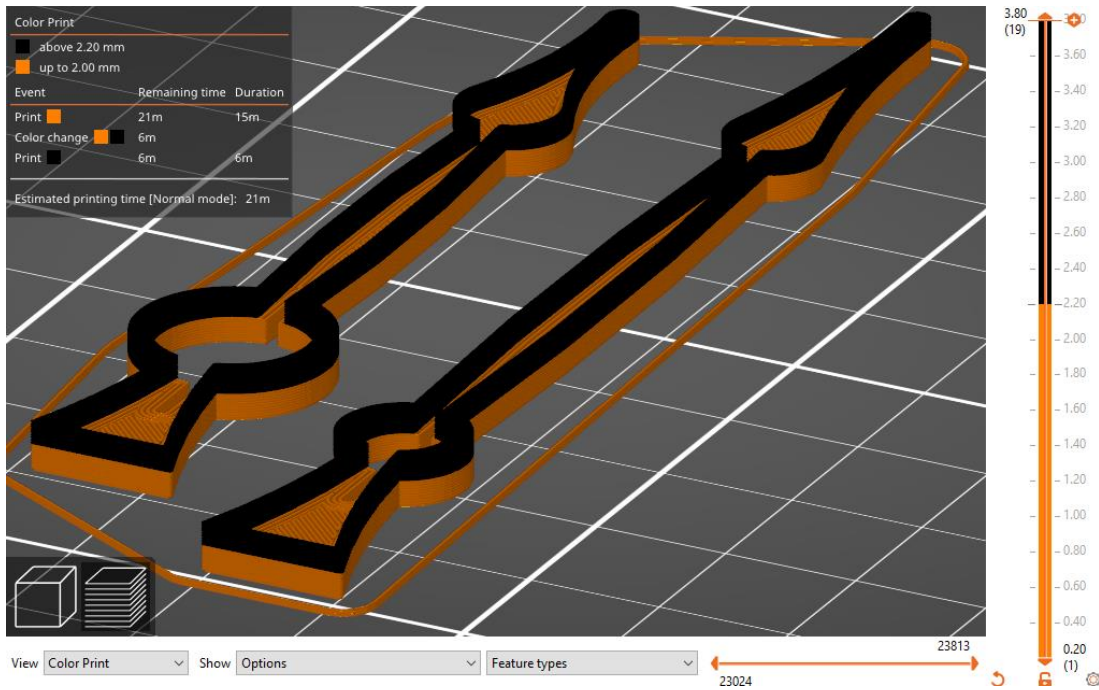


Figure 18: Color highlights on hands

Here is a diagram showing the various gears used in the clock.

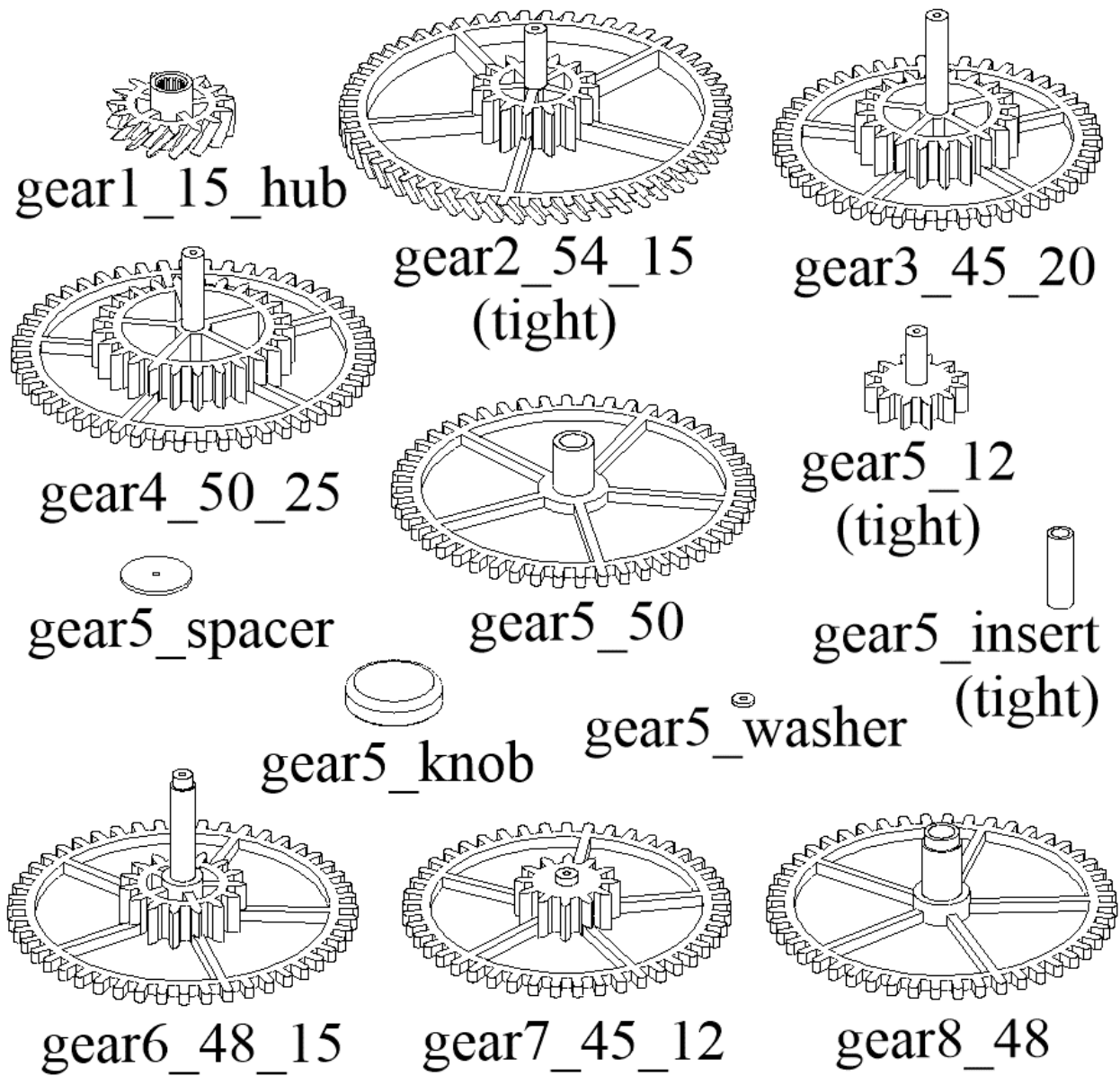


Figure 19: Gear reference diagram



## Building the Clock

Start by printing all the parts as described earlier. A few components need pre-assembly before being placed into the clock.

### Component Pre-Assembly

The gear 2 center hole needs to hold firmly onto the shaft so the second hand rotates with the gear. It doesn't need to be super tight, just snug enough to hold. The printed center hole is undersized and may need to be enlarged if it is too small. It is often sufficient to manually turn a 1/16" or 1.5mm drill bit through the center hole. If using a power drill, go slowly with one light pass. This often leaves just enough material to hold on to the shaft.

Gently tap the 4.375" shaft into the center hole of gear 2. The end should stick out the bottom around 0.4". Not much force should be needed. Glue it in position if the hole is too loose.

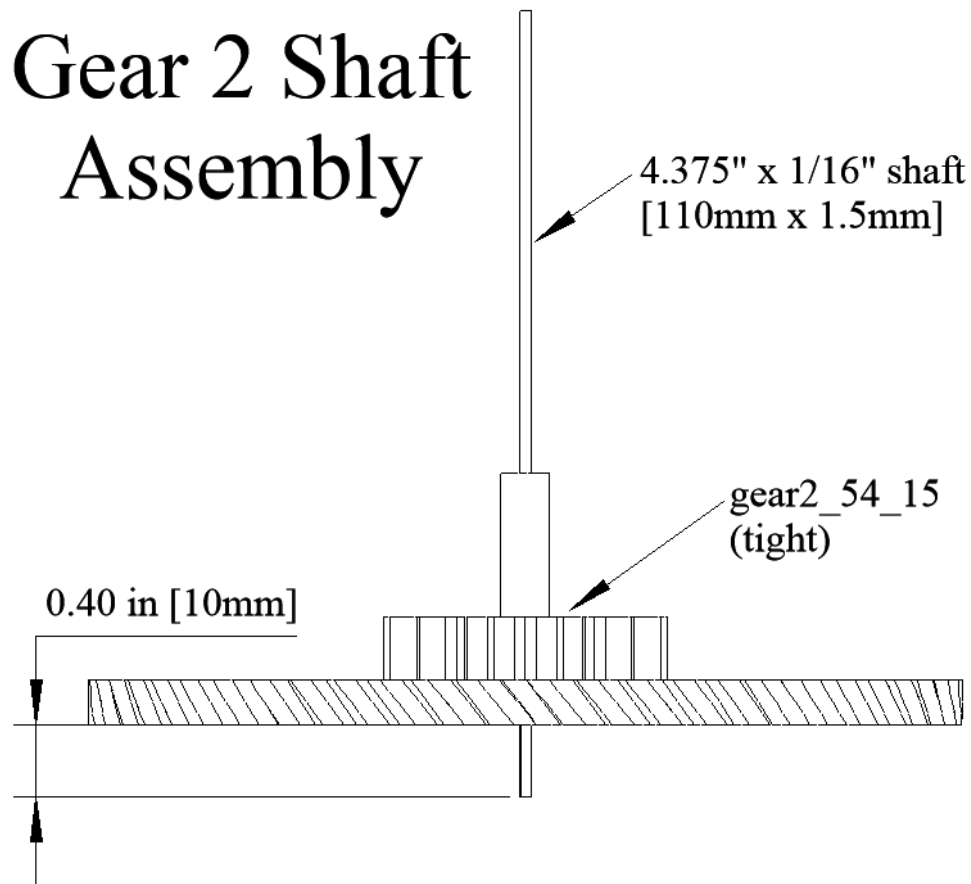


Figure 20: Gear 2 shaft assembly

The gear 5 assembly incorporates a friction clutch to hold when the clock is running and slip when setting the time. Gear5\_insert and gear5\_12 are tight on the 3.5" shaft and everything in between is allowed to rotate. The pen spring provides a slight amount of pressure.

Slightly drill out the two tight components (gear5\_insert and gear5\_12) if needed. Gently tap the 3.5" shaft onto gear5\_12 until it sticks out around 0.3". The remaining components can be added and gear5\_insert should be placed over the pen spring. Gently tap gear5\_insert until it is nearly bottomed out. The large and small gears should be able to rotate independently.

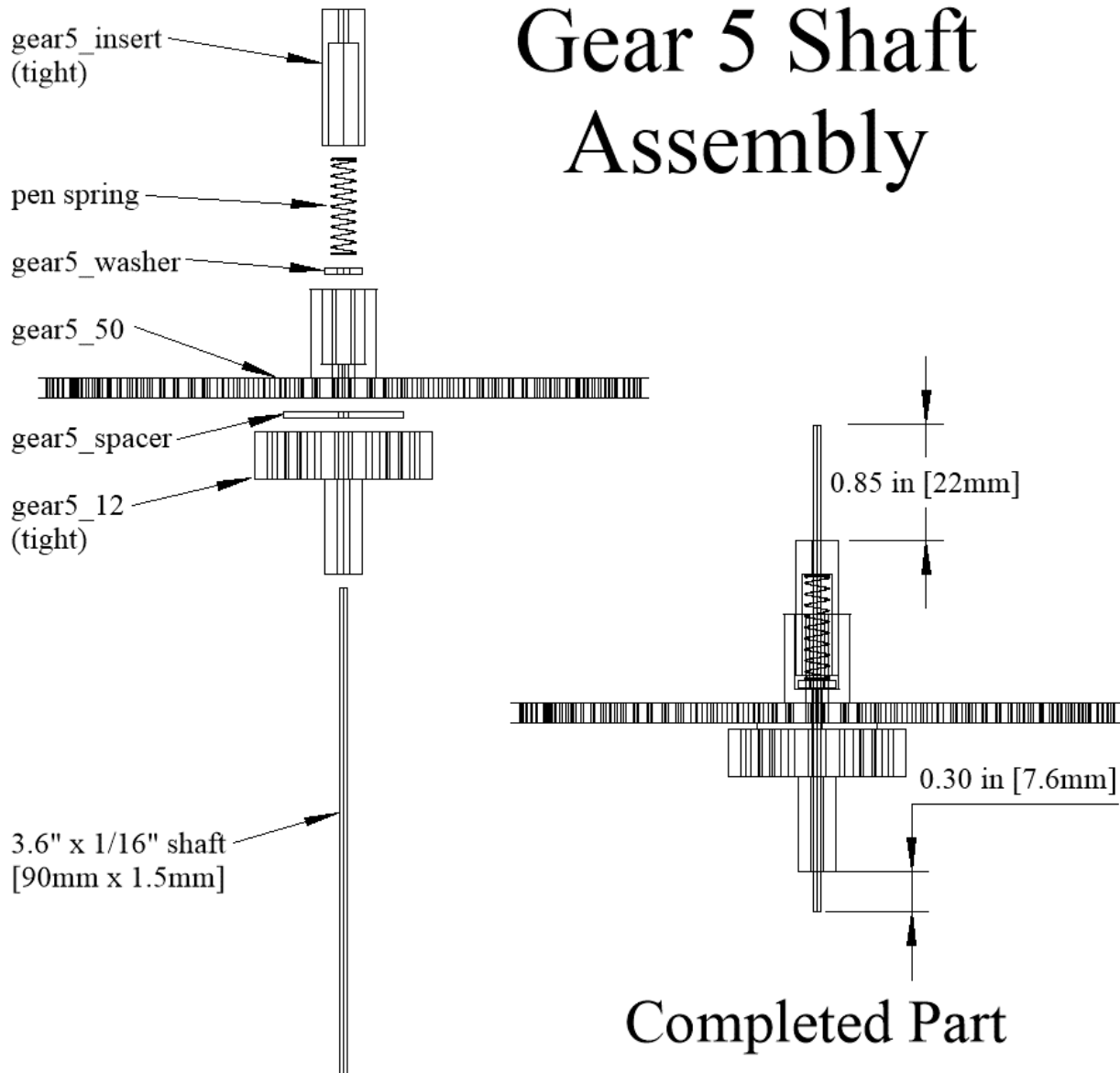


Figure 21: Gear 5 shaft assembly

The frame is provided with two print options. Large printers should print the full size frame. Printers smaller than 205x160mm will need to print the segmented frame and assemble the pieces as shown below. Alignment rods and screws hold everything together. The segmented frame is as good as the integrated frame, but it will need a few extra components.

The back frame assembles using two 1/16" by 3" shafts and a single 6x3/4" wood screw.

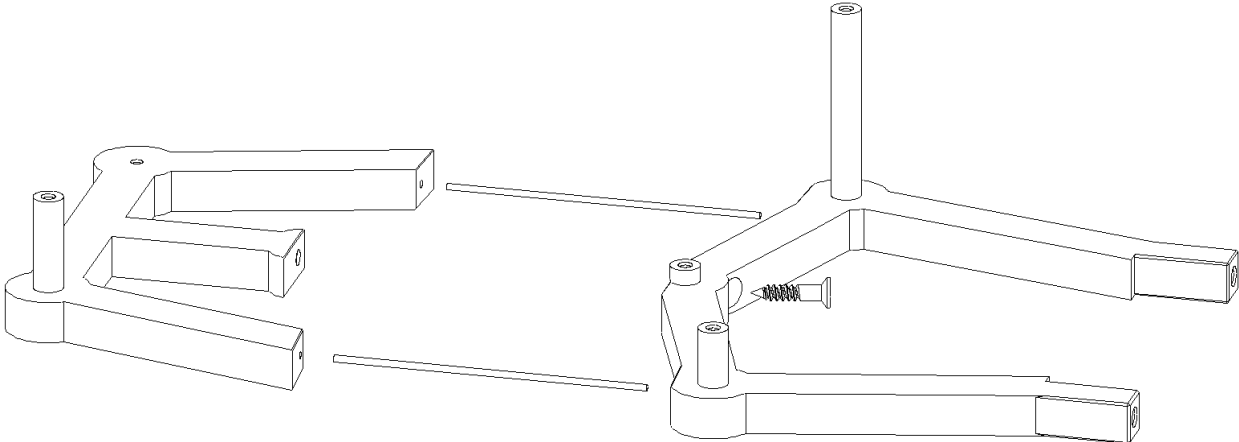


Figure 22: Segmented back frame assembly

The front frame uses four 1/16" by 3" long shafts and two 6x3/4" wood screws. These two screws will be visible on the sides of the clock.

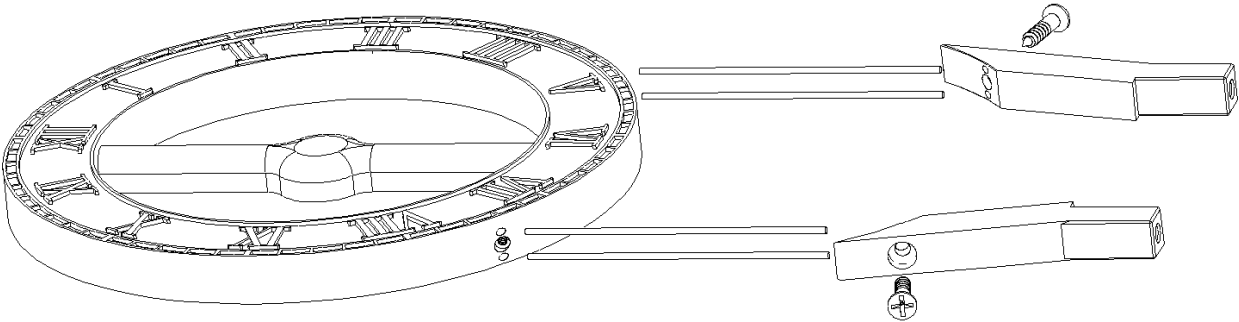


Figure 23: Segmented front frame assembly

Print gear1\_15\_hub and attach it directly to the stepper motor if you have the motor with a 14 tooth spline. It may be a tight fit, but should eventually slide on with pressure. The helical gear teeth will hold it in place if it is a loose fit.

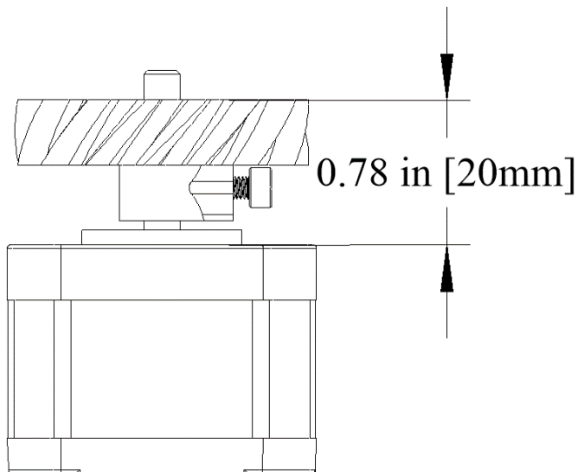


Figure 24: Stepper motor gear position

Motors with a 5mm shaft should use gear1\_15\_screw. A small M3x10mm screw holds the gear to the shaft. Any head style should work. Socket head M3x10mm screws are often used to build 3D printers, so you may have some screws in your spare parts kit.

It is OK for the shaft to stick through the end of the gear. The top of the gear should be about 0.78" (20mm) above the motor body.

### Checking Component Fit

It is a good idea to check the fit of the remaining components before assembling the clock.

Step 1) Check that arbors fit in their respective holes. The 1/16" or 1.5mm arbors need to fit into the holes in the frame. The arbors should also pass through the large gears. Drill them out if needed. It is desirable for the holes to be just large enough for the arbor to pass through and spin, so don't enlarge the holes too much. Gear2\_54\_15, gear5\_12, and gear5\_insert need to be tight on the shaft, so don't drill them loose. These parts should have already been assembled in the previous section.

Step 2) Check that the minute hand (gear6\_48\_15) and hour hand (gear8\_48) gears fit into each other and into the front frame. Gear 6 needs to fit inside gear 8 and gear 8 needs to fit inside the center hole in the front frame. Rolled up sandpaper or a round file can be used to enlarge the inside holes if needed. Or wrap sandpaper around the outer shafts to reduce the diameters slightly.

Step 3) Check that the completed Arduino and motor controller fits into the base. It should go in at an angle until the mini USB port is lined up, then it drop into a flat position.

Step 4) Check that the frame fits into the base. The pegs are tapered so they should go in easily at the start and get tighter when they bottom out. Sand the pegs slightly if needed. The bottom screws should pull everything together. Use the screws to help push the frame out if you need to take the clock apart.

## Adding the Gears

Add the previously completed gear 2 assembly into the center hole of the back frame.

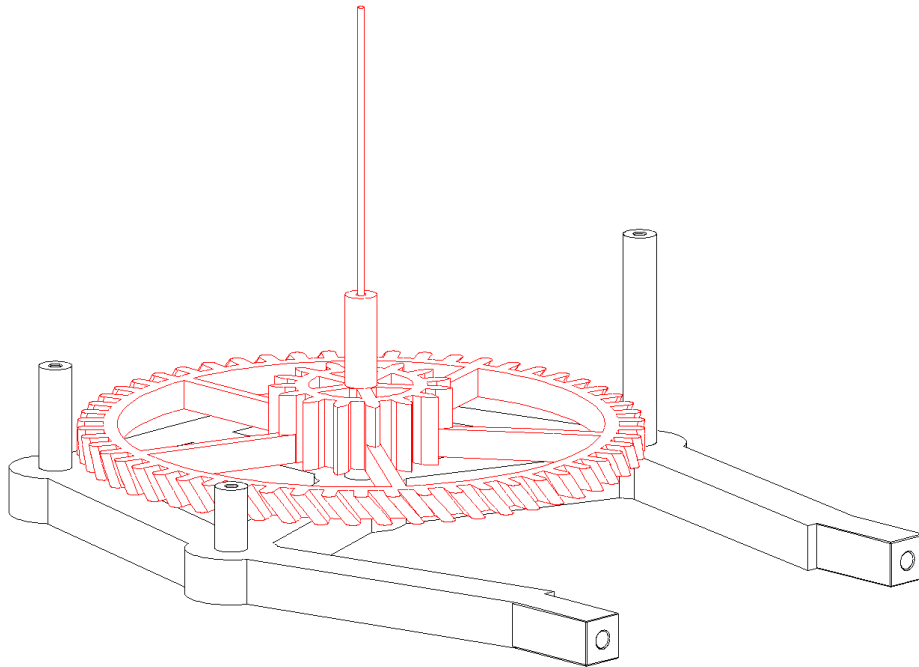


Figure 25: Add gear 2 assembly

Add a 1/16" by 3" shaft and gear3\_45\_20 into the lower left frame position. It should mesh with gear 2.

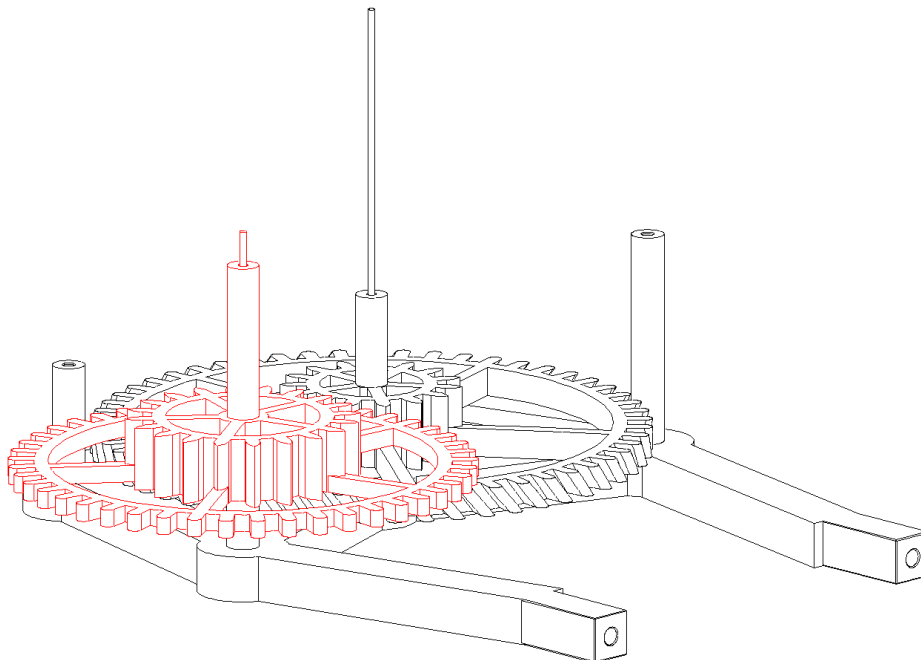


Figure 26: Add gear 3

Add a 1/16" by 3" shaft and gear4\_50\_25 into the upper left position. It meshes with gear 3.

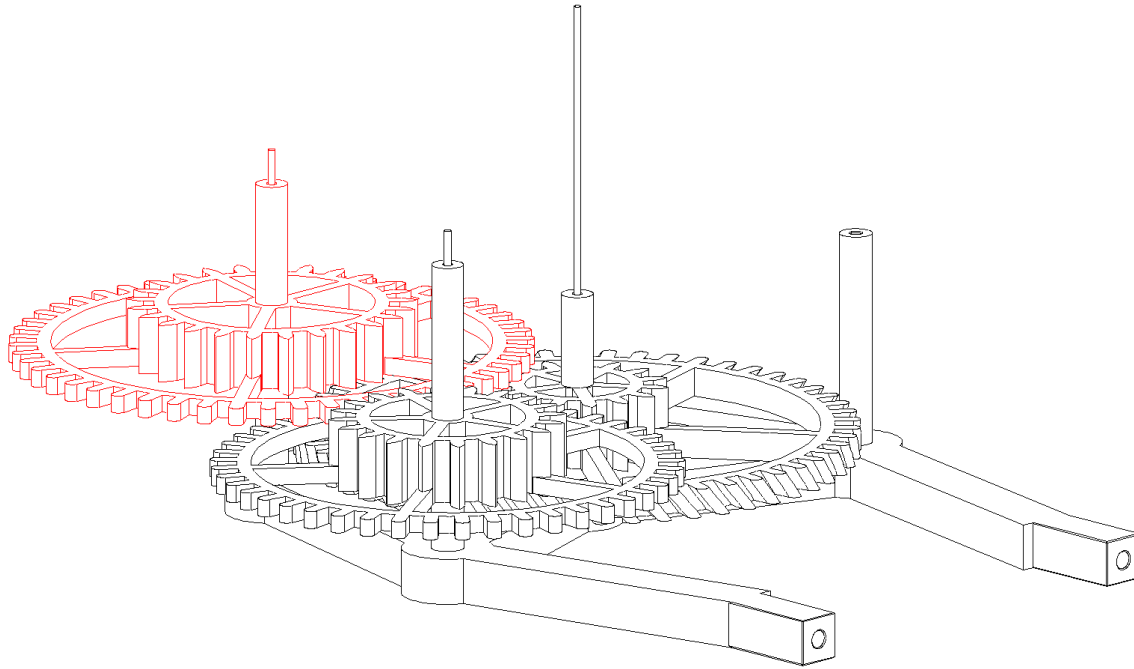


Figure 27: Add gear 4

Add the previously completed gear 5 assembly in the upper right position. The arbor sticks through the back of the frame.

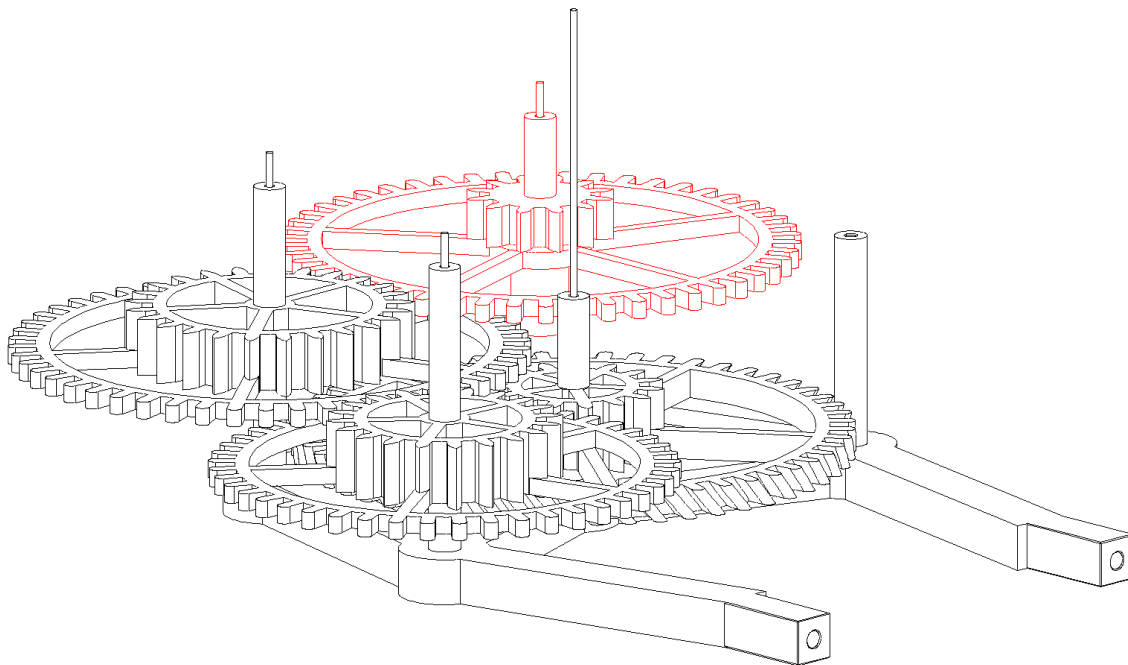


Figure 28: Add gear 5 assembly



Add gear5\_knob to the arbor using a M3x10mm screw where the gear 5 arbor sticks through the back of the frame. It may help to grind a small flat on the shaft for the screw to hold better. Here is a view from the back of the clock.

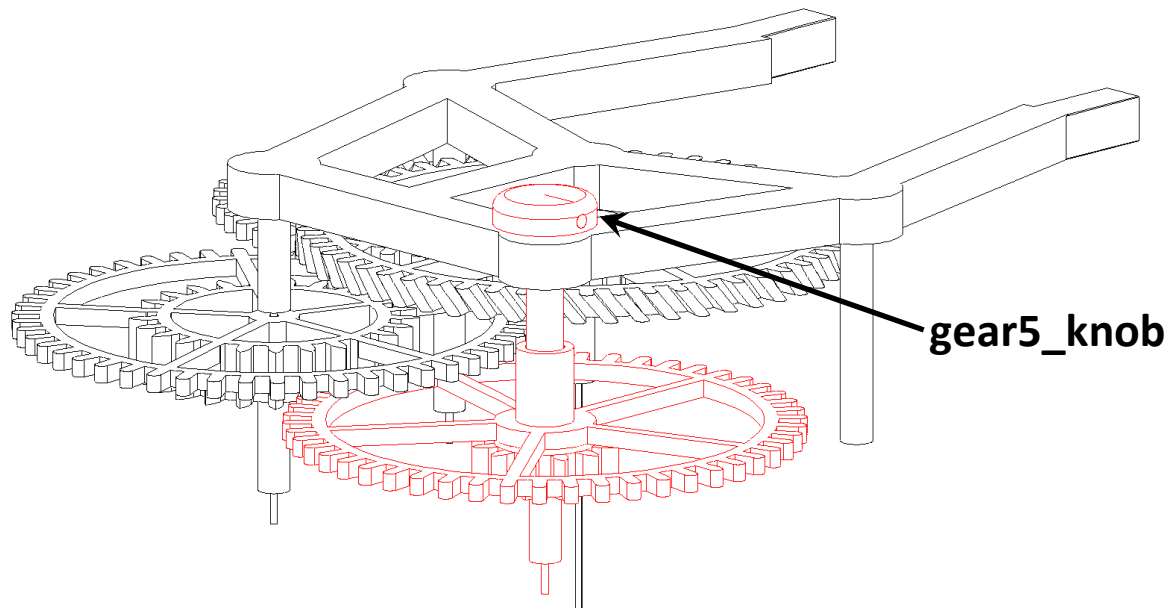


Figure 29: Knob added to gear 5 behind the clock

Add gear6\_48\_15 to the gear 2 arbor in the center of the clock. This gear drives minute hand and will pass through the front frame. It meshes with gear 5.

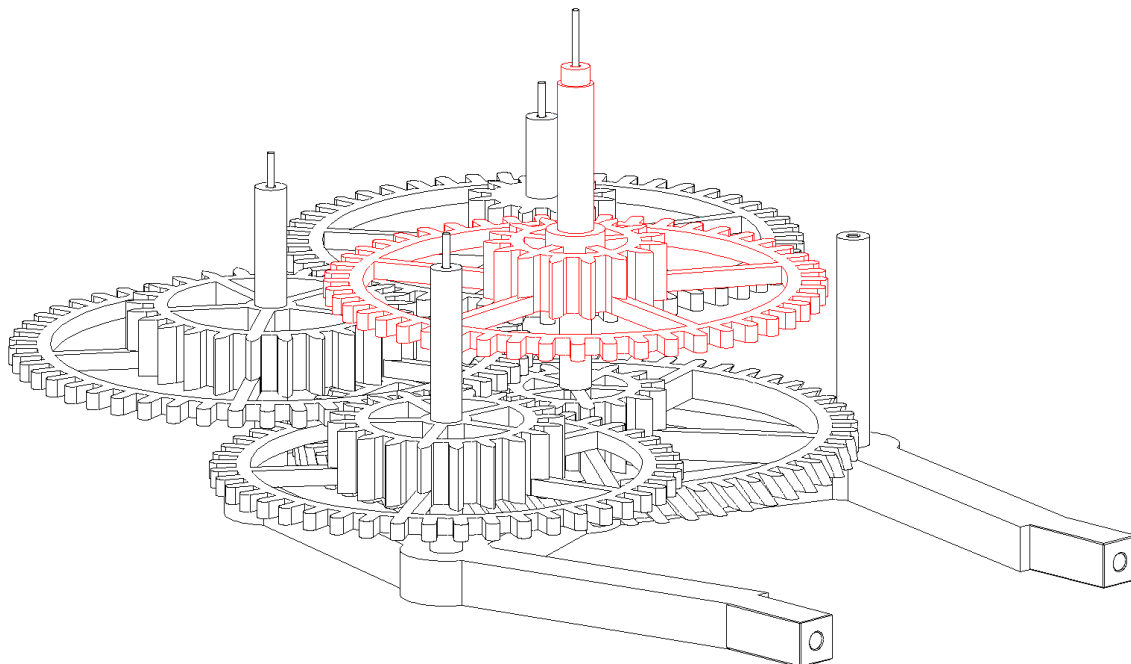


Figure 30: Add gear 6

Add a 1/16" by 3" arbor and gear7\_45\_12 in the lower right position. It meshes with gear 6.

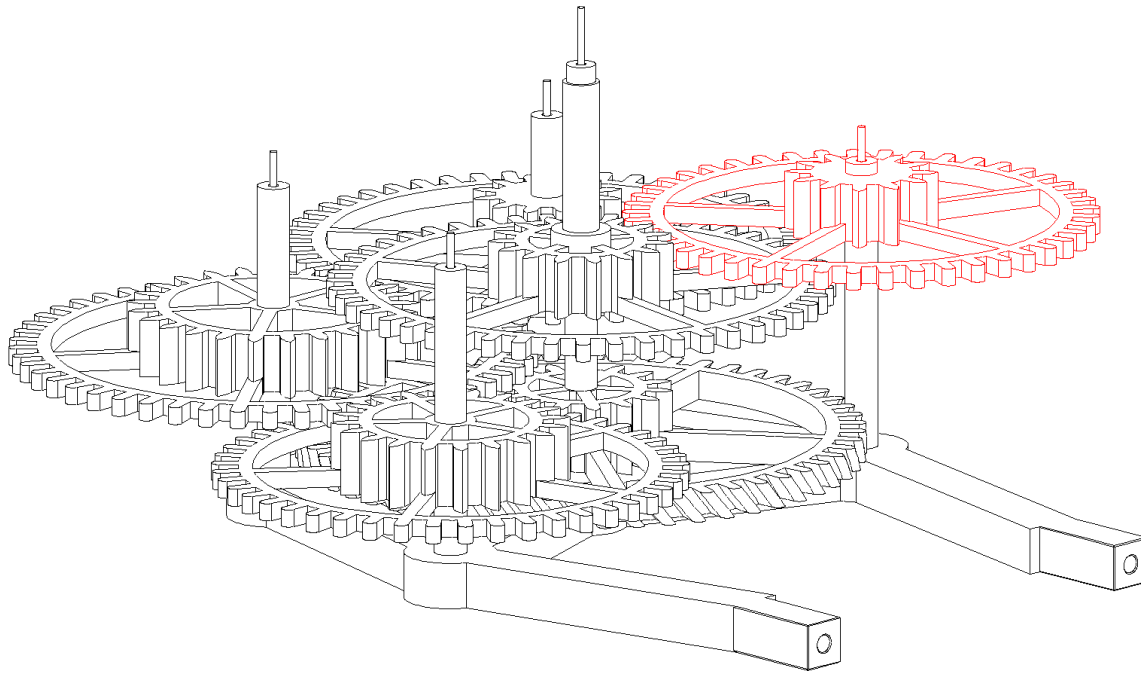


Figure 31: Add gear 7

The last remaining large gear is gear8\_48 that goes on the gear 2 arbor in the center of the clock. It meshes with gear 7 and passes through the front frame for the hour hand.

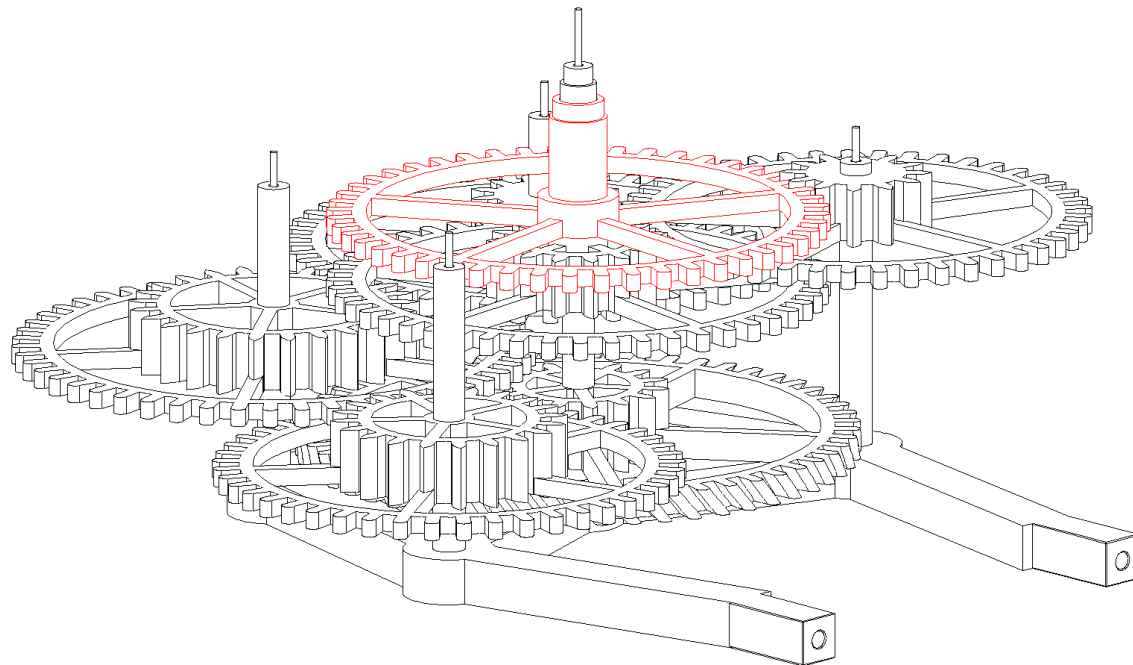
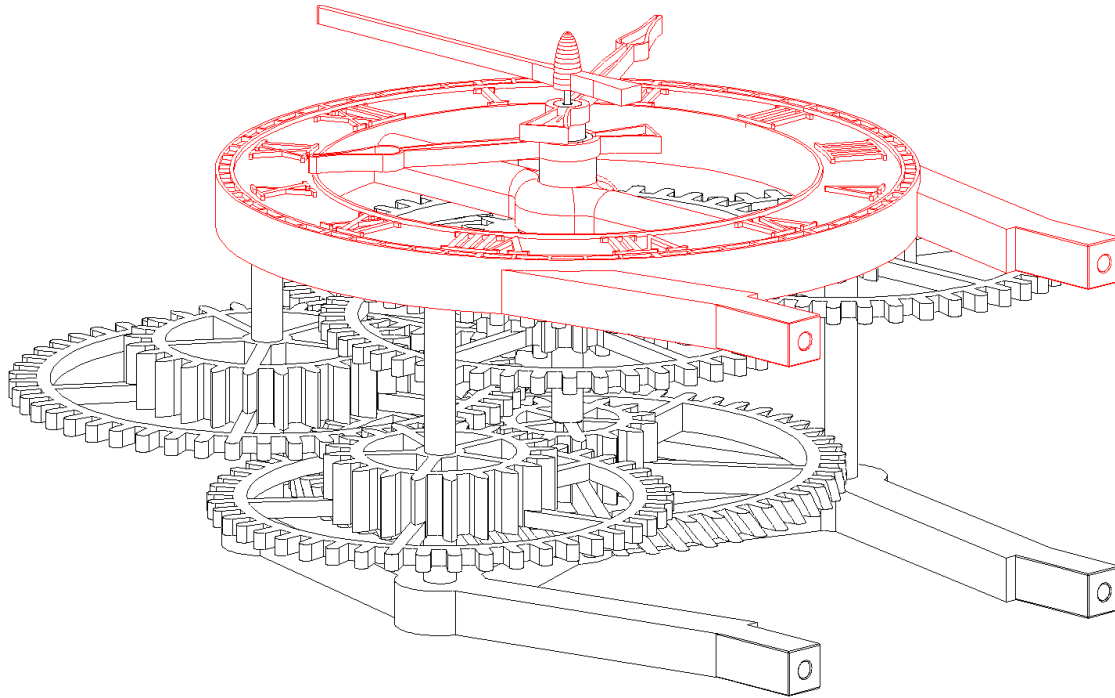


Figure 32: Add gear 8

Add the front frame onto the clock. Start by placing the dial over the central arbor. Then look from the side and align the remaining arbors with their respective hole. The holes in the front frame are slightly chamfered to make it easier. The frame should drop into place as long as the arbors are close to the proper position. The hands can be added later.



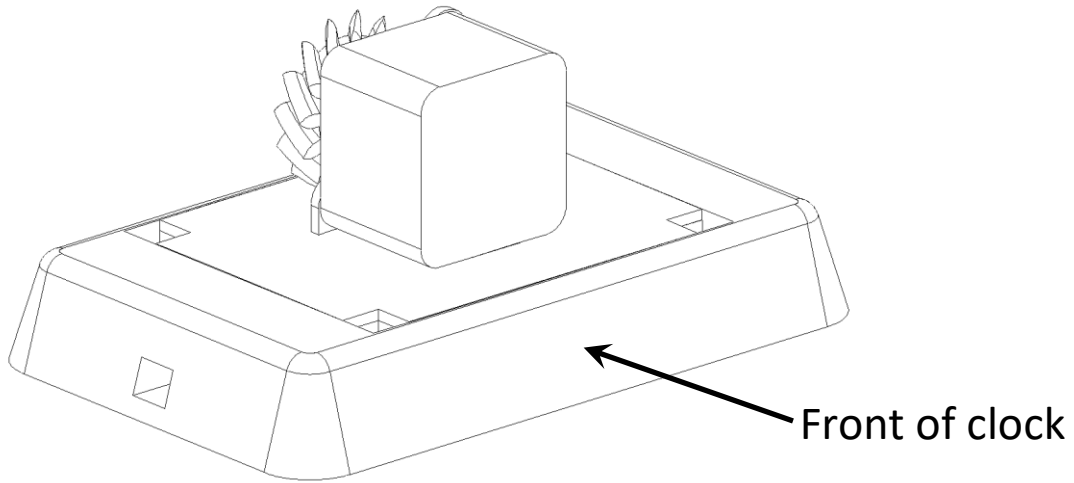
*Figure 33: Add the front frame*

Attach the stepper motor to the motor mount using two M3x10mm screws. These screws may be available in the spare parts kit if you built your own 3d printer.

Feed the stepper cable through the wiring hole in the motor mount. The hole is sized to allow the motor controller to fit through if you have a motor without a connector. You will need to remove the Arduino Nano from the motor controller to make it fit. I will post the CAD files for the base if you want to modify the size of the hole. Or message me with dimensions if you want me to make the edit.

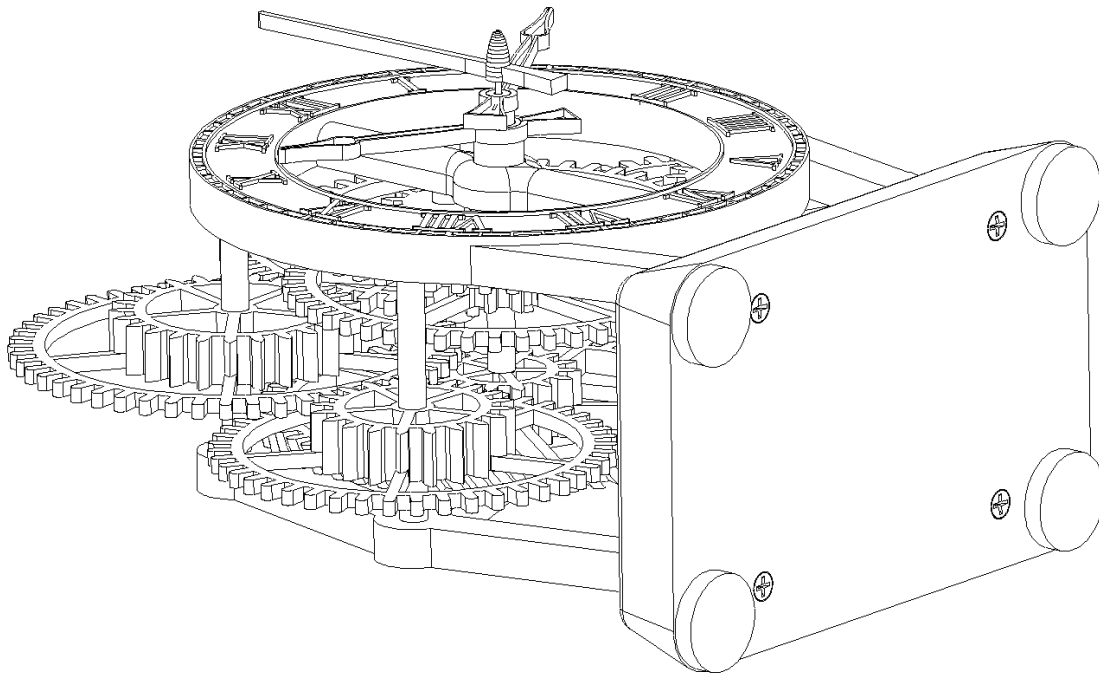
Attach the stepper motor to the Arduino Nano and motor controller circuit. Place the Arduino module into position in the base with the motor control circuit on top. Start by lining up the USB connector in its hole, then tilt everything into place. Wrap the wires so the motor mount can drop into position without pinching the wires. The gear should be positioned towards the back of the clock. If you are using the base with side power, rotate the motor mount so the power cord will be facing the proper direction. The base is symmetrical so it can be rotated 180 degrees.

Here is what the base and motor assembly would look like with the power on the left side.



*Figure 34: Motor and electronics added to base*

Add the clock assembly onto the base and attach with four 3/4" wood screws. Four optional 1" diameter felt pads fit perfectly on the bottom corners of the base.



*Figure 35: Fully assembled clock*

## Calibration

The clock is now fully assembled and nearly ready to run. Plug in the mini USB cable and the second hand should start rotating if the Arduino has been programmed. The final step is to calibrate the clock by changing a few variables in the program.

If the second hand rotates backwards, then change

```
#define DIRECTION 1  
to  
#define DIRECTION -1
```

The motor speed depends on the accuracy of the Arduino Nano oscillator. They supposedly use ceramic resonators which can have an error of up to +/-0.5%. The good news is that they appear to be stable over time, so the clock will be reasonably accurate once the initial tolerance is accounted for. The algorithm uses three variables for calibration. The exact values needed for your clock may be different.

```
#define MAJOR_DELAY 331  
#define MINOR_DELAY 250  
#define MICRO_DELAY 10
```

MAJOR\_DELAY sets the coarse time delay of the clock. This delay in microseconds gets added 180,000 times per minute. Adding one digit makes the clock run by about 10.84 seconds slower per hour. Subtracting a digit makes the clock run 10.84 seconds faster.

MINOR\_DELAY is used for fine tuning the clock. It is added 180 times per minute, so it has 1/1000<sup>th</sup> as much effect as MAJOR\_DELAY. Each adjustment of the value changes the rate of the clock by 0.01084 seconds per hour or 0.260 seconds per day. The acceptable range for this variable is 10 to 1010. Increase the value to slow the clock down slightly. If the value gets above 1010, then reset it back to 10 and add one to MAJOR\_DELAY.

MICRO\_DELAY is used for very tiny adjustments. It has 1/100<sup>th</sup> as much of an affect as MINOR\_DELAY, only adding a small delay 1.8 times per minute. Each change affects the clock be around 0.078 seconds per 30 days. I am not really sure if the clock needs this much accuracy, but the additional overhead is very minor.

Set the time on the clock by rotating the knob behind the frame. Adjust it to match a stable clock source like a cell phone. It is OK to hold gear 2 to halt the second hand movement until it matches the real time. Let the clock run for 24 hours or longer and see how much it has drifted. Adjust MAJOR\_DELAY up or down to get the clock close to the final accuracy. You may need to let the clock run for a few days to adjust MINOR\_DELAY, and a month or two to set MICRO\_DELAY.

Each clock will need to be calibrated separately and the values may be different. If you build multiple clocks, you will need to record the settings for each clock. I do not know how to read back the values that have been programmed into a clock, so you will need to write down the values. You can include duplicate sections in the code and comment out all but one for each clock.

## Appendix A: Motor Control Experiments

Finding a good motor for this clock was a big challenge. I tried many different types in hopes of making the clock run quietly. AC servo motors are used to power old time punch machines, but they seem to be getting hard to find. They are also relatively expensive as the old stock gets used up. Some have built in gear boxes to slow down the output to one revolution per hour. Most run directly from the main power



Figure 36: AC synchronous motor

supply, which is a bit risky to recommend for others to build. Even one person getting shocked when building the clock is too much. I also don't want to release a design and frustrate builders because there are no more motors available. I decided to keep looking for a more common motor running at a lower voltage.

AC synchronous motors listed as microwave oven motors seem popular on eBay with outputs turning at 5-6RPM. Some motors run at 120V or 240V, but 12VAC or 24VAC motors are also available. Power is typically 3-4W at 12VAC, so they get a little warm, but power can be reduced to around 1W by running them at half the nominal voltage. There is still more than enough torque to run a small desk clock. There were just a few minor problems to solve. The first issue is that the rotational direction is not always the same. They randomly rotate in any direction at powerup.

I designed a simple gear box to solve the rotational problem and posted it to [One way gear by StevePeterson - Thingiverse](#). An even simpler solution is to add a ratchet to prevent it from rotating in the wrong direction. The motor can be coerced to rotate the desired way.

The next problem is that the rotational speeds are not exactly as advertised. The first two brands I tried had slightly different speeds. Both were labeled as 5-6RPM, so they were expected to rotate at 6RPM on 60Hz AC power, but one turned at 5.824RPM and the other turned at 6.228RPM. The internal gearing of each was different enough that it would be too difficult to recommend this style of motor for a clock.



Figure 37: Microwave oven synchronous motors



2PCS- 24BYJ48 Stepper Motor

DC 12V

Reduction ratio: 1:64



Figure 38: Small geared stepper motors

I kept searching for a better clock motor. DC gear motors were ruled out because of too much speed variation to make a reliable clock. Adding sensors would make the design too complicated.

Another solution is to use a stepper motor and directly control the rotational speed. I tried the common (and really cheap) geared stepper motor shown on the left. These are simple 5 wire stepper motors that can be driven using an Arduino and a ULN2003 driver chip. It worked OK, but only for a short time. The first test clock developed a ticking noise within a day of operation. It seems like the internal gears are too weak to stand up to the very light load of a clock when run continuously. They appear to be designed for intermittent use.

My next set of experiments used a simple stepper motor. One early experiment used a 4 wire stepper motor directly connected to a low voltage AC transformer with a capacitor to delay the signal to the second coil. This lets it run as a synchronous motor. The control circuit is incredibly simple with just a transformer and a capacitor. The motor turned, but the rotational speed was quite fast at 1.2 times per second at 60Hz or 1.0 times per second at 50Hz. Gearing would need to be added to slow it down. Gears running this fast will be noisy and the motor had a loud 120Hz hum. The audible noise was too much to be useful in a clock.

The final solution was to continue using a stepper motor and develop the motor control circuit described in this document. NEMA17 motors were selected as the smallest size that is reasonably common. These motors should be extremely robust. One set of gears are used between the motor and the second hand. This allows the motor to be positioned low where much of the motor is hidden behind the frame.

The clock appears to be very stable after the initial calibration. It seems to be accurate to within a few seconds per week, although there is a small concern regarding temperature stability. I placed the clock in the freezer overnight and the clock gained a little under 1 second per hour. It should hold an accuracy of a few seconds per day when operating within the normal temperature variation with heating and air conditioning. A long term plan is to add a real time clock to provide a time reference so the clock can be continuously calibrated across temperature variations.

## Appendix B: Arduino Nano Algorithm

Here is the complete algorithm used to drive the clock. Cut and paste into the Arduino IDE. Updated versions of the code may be created, although I am not yet sure where they will be posted.

```
/*
 * stepper driver attempting smooth positioning
 * using multiple semi-binary resistor values
 * and a lookup table for sine wave current levels
 *
 * Revision 1.0 30-Aug-21 Initial release
 *
 * The lookup table has 250 entries per quadrant
 *
 * Total number of steps per full cycle is 250 * 4 = 1000
 *
 * Total steps per revolution is 1000 * 50 = 50000
 *
 * Uses 5 resistors per port plus 3 sink resistors
 * PB4:0 with 5 resistors driving into PC2:0 with 3 resistors
 * PD6:2 with 5 resistors driving into PC5:3 with 3 resistors
 *
 * Driving resistors are 100, 220, 470, 1000, and 2200 ohms
 * The 5 resistor values selected are reasonably monotonic for all values
 * Sinking resistors are 3X 100 ohms in parallel
 * Stepper motor is 30-35 ohms (very important)
 *
 * Full code routine has 1000 steps (DIRECTION = -1)
 *   Steps: portD portB portC DirD DirB
 *   init: 0x00 0x00 111111 31 0
 *   0-249: 0x00 0x00 111111 31->0 0->31
 *   250-499: 0x7c 0x00 000111 0->31 31->0
 *   500-749: 0x7c 0x1f 000000 31->0 0->31
 *   750-999: 0x00 0x1f 111000 0->31 31->0 (step 999 starts over as step 0)
 *
 * Full code routine has 1000 steps (DIRECTION = 1)
 *   Steps: portD portB portC DirD DirB
 *   init: 0x00 0x1f 111000 31 0
 *   0-249: 0x00 0x1f 111000 31->0 0->31
 *   250-499: 0x7c 0x1f 000000 0->31 31->0
 *   500-749: 0x7c 0x00 000111 31->0 0->31
 *   750-999: 0x00 0x00 111111 0->31 31->0 (step 999 starts over as step 0)
 */

////////////////////////////////////
// Settings used to calibrate the clock //
////////////////////////////////////
//
// MAJOR_DELAY is used 1000 times per cycle (for setting coarse delay)
// Adding one step to MAJOR_DELAY slows the rate around 10.84 seconds per hour
//
// MINOR_DELAY is used only once per cycle (for fine tuning overall loop time)
// Adding one step to MINOR_DELAY slows the rate around 0.01084 seconds per hour
// Keep this value within the range of 10 to 1000
// If it gets above 1000, reset to 10 and add 1 to MAJOR_DELAY
//
// MICRO_DELAY is used once every 100 cycles (for really minor fine tuning)
// Adding one step to MICRO_DELAY slows the rate around 0.078 seconds per 30 days
// Keep this value within the range of 10 to 110
// If it gets above 110, subtract 100 and add 1 to MINOR_DELAY

#define MAJOR_DELAY 331
#define MINOR_DELAY 393
#define MICRO_DELAY 10

// direction can be changed by setting the following variable to 1 or -1
#define DIRECTION -1
```

```

void setup() {
  // put your setup code here, to run once:
  PORTB = 0x00;
  if (DIRECTION == 1) {
    PORTB = 0x00;
    PORTD = 0x00;
    PORTC = 0x3f;
  } else {
    PORTB = 0x1f;
    PORTD = 0x00;
    PORTC = 0x38;
  }
  DDRB = 0x20; // portB starts off (except for status LED)
  DDRD = 0x7c; // portD starts on
  DDRC = 0x3f; // portC is always on

  // turn off ALL internal pullups
  MCUCR |= _BV(PUD);
}

void loop() {
  // put your main code here, to run repeatedly:

  int i; // minor loop step counter
  int cycle_count = 0; // counter used to add MICRO_DELAY every 100 cycles

  // lookup table optimized for 100, 220, 470, 1000, and 2200 ohm resistors
  static unsigned char stepper_data[] = {0, 0, 0, 0, 0, 1, 1, 1, 1, 1,
    1, 1, 1, 1, 1, 1, 2, 2, 2, 2,
    2, 2, 2, 2, 2, 3, 3, 3, 3, 3,
    3, 3, 3, 3, 3, 4, 4, 4, 4, 4,
    4, 4, 4, 4, 5, 5, 5, 5, 5, 5,
    5, 5, 6, 6, 6, 6, 6, 6, 6, 7,
    7, 7, 7, 7, 7, 7, 7, 7, 8, 8,
    8, 8, 8, 8, 8, 8, 8, 9, 9, 9,
    9, 9, 9, 9, 10, 10, 10, 10, 10, 11,
    11, 11, 11, 11, 11, 11, 12, 12, 12, 12,
    12, 12, 12, 13, 13, 13, 13, 13, 13, 14,
    14, 14, 14, 14, 15, 15, 15, 15, 15, 15,
    16, 16, 16, 16, 16, 17, 17, 17, 17, 17,
    17, 18, 18, 18, 18, 18, 19, 19, 19, 19,
    19, 19, 19, 20, 20, 20, 20, 20, 20, 21,
    21, 21, 21, 21, 22, 22, 22, 22, 22, 22,
    23, 23, 23, 23, 23, 23, 23, 24, 24, 24,
    24, 24, 24, 24, 24, 25, 25, 25, 25, 25,
    25, 26, 26, 26, 26, 26, 26, 26, 27, 27,
    27, 27, 27, 27, 27, 27, 28, 28, 28, 28,
    28, 28, 28, 28, 28, 29, 29, 29, 29, 29,
    29, 29, 29, 29, 29, 29, 30, 30, 30, 30,
    30, 30, 30, 30, 30, 30, 30, 30, 30, 30,
    30, 31, 31, 31, 31, 31, 31, 31, 31, 31,
    31, 31, 31, 31, 31, 31, 31, 31, 31, 31};

  // main loop
  while (1) {

    // add additional minor delay (only once per cycle)
    delayMicroseconds(MINOR_DELAY);

    // check if MICRO_DELAY needs to be added (once every 100 cycles)
    if (cycle_count++ >= 100) {
      delayMicroseconds(MICRO_DELAY);
      cycle_count = 0;
    }

    // steps 0 to 249
    for (i = 0; i <= 249; i++) {
      DDRB = stepper_data[i]; // count portB 0 to 31
      DDRD = stepper_data[249 - i] << 2; // count portD 31 to 0
    }
  }
}

```

```

    delayMicroseconds(MAJOR_DELAY);
}

// changes needed between steps 249 and 250
if (DIRECTION == 1) {
    PORTC = 0x07;
    PORTD = 0x7c;
} else {
    PORTC = 0x00;
    PORTD = 0x7c;
}

// steps 250 to 499
for (i = 0; i <= 249; i++) {
    DDRD = stepper_data[i] << 2;          // count portD 0 to 31
    DDRB = stepper_data[249 - i];        // count portB 31 to 0
    delayMicroseconds(MAJOR_DELAY);
}

// changes needed between steps 499 and 500
if (DIRECTION == 1) {
    PORTC = 0x00;
    PORTB = 0x1f;
} else {
    PORTC = 0x07;
    PORTB = 0x00;
}

// steps 500 to 749
for (i = 0; i <= 249; i++) {
    DDRB = stepper_data[i];              // count portB 0 to 31
    DDRD = stepper_data[249 - i] << 2;  // count portD 31 to 0
    delayMicroseconds(MAJOR_DELAY);
}

// changes needed between steps 749 and 750
if (DIRECTION == 1) {
    PORTC = 0x38;
    PORTD = 0x00;
} else {
    PORTC = 0x3f;
    PORTD = 0x00;
}

// steps 750 to 999
for (i = 0; i <= 249; i++) {
    DDRD = stepper_data[i] << 2;          // count portD 0 to 31
    DDRB = stepper_data[249 - i];        // count portB 31 to 0
    delayMicroseconds(MAJOR_DELAY);
}

// changes needed between steps 999 and 0
if (DIRECTION == 1) {
    PORTC = 0x3f;
    PORTB = 0x20;          // status LED is turned on
} else {
    PORTC = 0x38;
    PORTB = 0x3f;          // status LED is turned on
}

} // end main loop
} // end program

```

## Appendix C: Some of my Other Clock Designs

Here are a few of the other clocks I have built. Many of them will eventually be released for others to build. The first is a grasshopper escapement to replace the deadbeat escapement in my one of my earlier clocks. It needs a bit of fine tuning before it can be released. The second image is a rendering of one of my designs as it may look after porting to use wooden gears.



Figure 39: Grasshopper clock and wood clock rendering



These are some sample wooden gears cut from solid wood using a new method to prevent expansion from humidity changes. They will eventually be used to create the rendered clock on the previous page.



Figure 40 Wooden gear experiments

Here are some of the prototypes before settling on the final design in this release. The largest size is physically impressive, but the larger gears are noisy. It took a lot of fiddling before finding a motor control circuit that was quiet enough to release.



Figure 41 Desk clocks



Here is the clock that started it all. It is posted to <https://www.thingiverse.com/thing:3524448>



Figure 42 Original Thingiverse design

This is my second clock posted to <https://www.myminifactory.com/object/3d-print-137009>



Figure 43: Large pendulum clock

A clock posted to <https://www.myminifactory.com/object/3d-print-32-day-clock-easy-build-156759> with a runtime up to 32 days between winding. It is also my easiest to build clock. Some of the features making it easier to build also make it more efficient so the runtime was increased to 32 days.

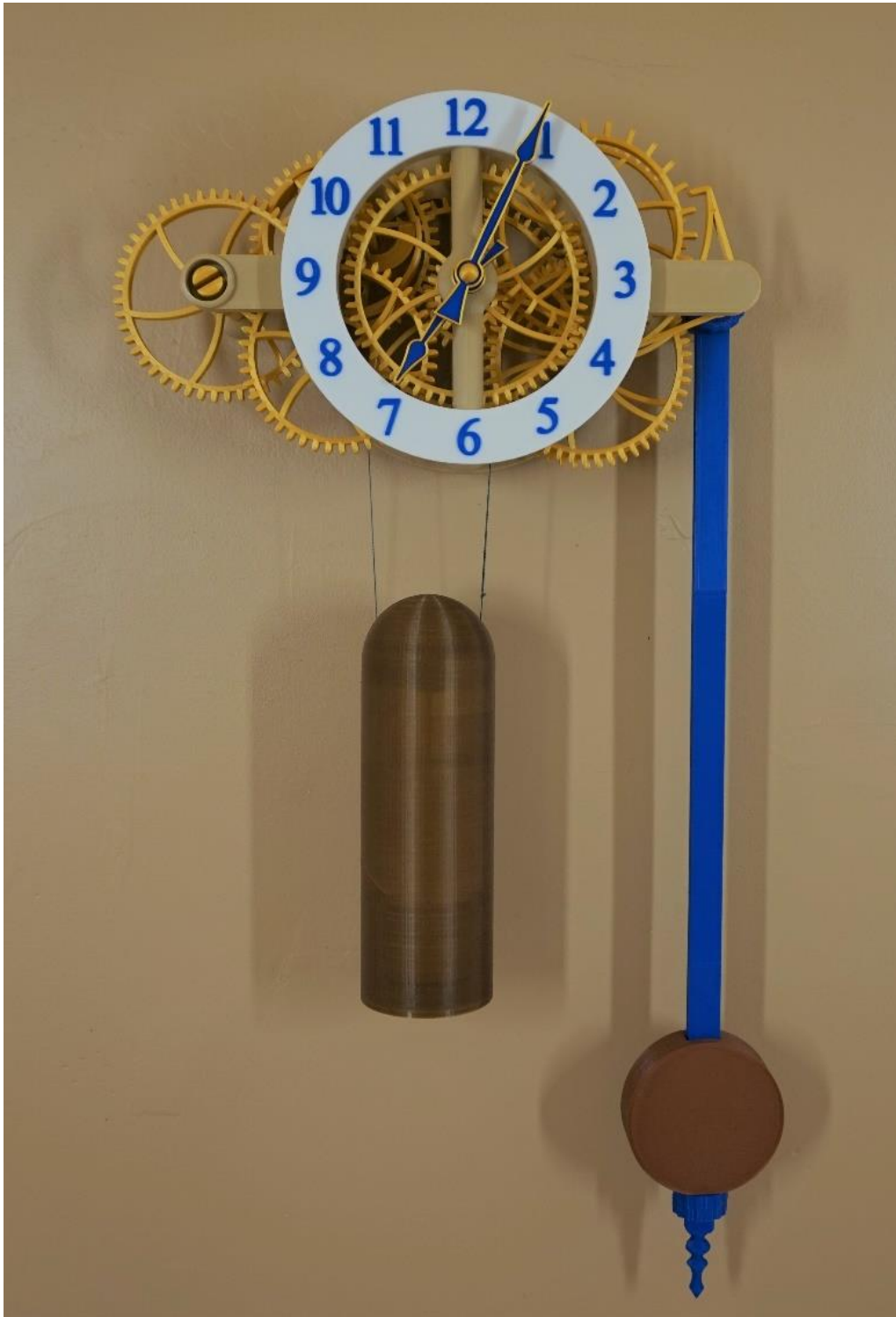


Figure 44: Easy build clock with 32 day runtime

An experiment for a potential future clock using an electromagnetic pendulum drive.



Figure 45: Electric pendulum drive