Flyback Chronograph

CSCE 462 - Fall 2023

Allen Li, Vivek Amarnani

Overview

- Goal
- Design Process
- Hardware
- Software
- Outcome and Improvement

Design Process

Plan + Familiarize

- Plan hardware needed and which software libraries to use
- Begin testing components to ensure functionality and controllability
- Test to see if hardware is up to par

Implement Functionality

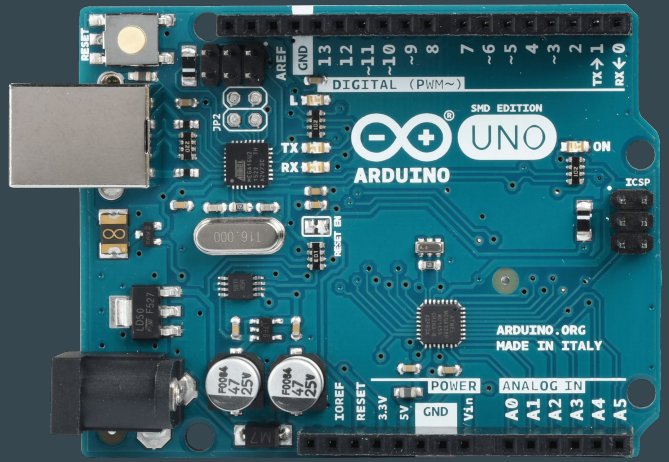
- Seconds motor needs to complete one full rotation every minute
- Hour/minute motor rotates 16 times to elapse 12 hours
    - One hour is 16/12 (1.33 rotations - 266.67 steps)
    - Control motor so that every minute is 266/60 steps
- RTC time for time-keeping
- Mode switching
- IR receiver and remote used to switch between modes

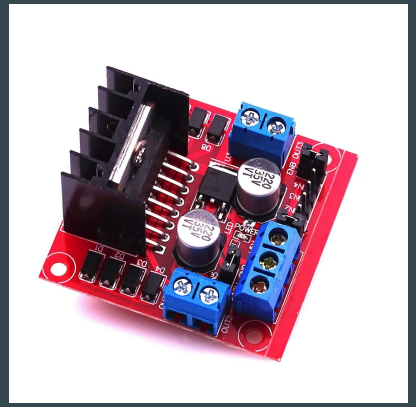
Put Together

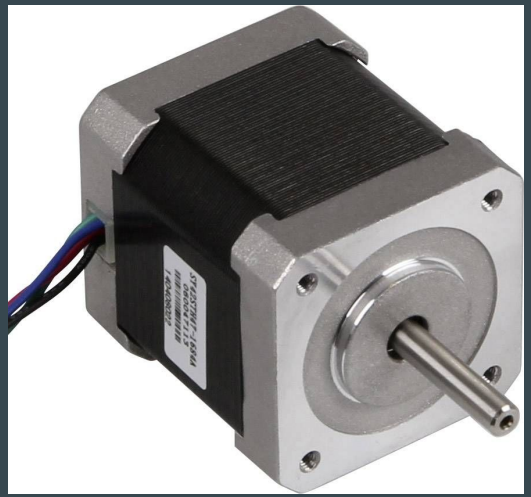
- Motors know their position and how to return back to 0 quickly to switch modes
- When turned on, the RTC time is reflected on the clock when in time-keeping mode
- The clock is aesthetically pleasing, highly legible, and retains all functionality

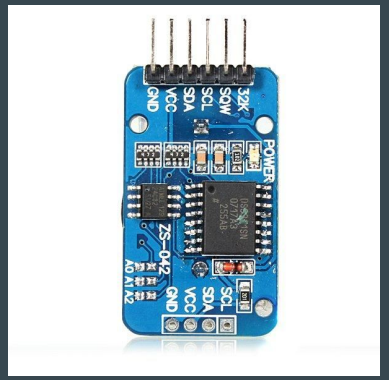
Hardware Parts Used

- Arduino UNO
- 2 - NEMA 17 Stepper Motors
- 2 - L298 Motor Drivers
- 1 - Clock gear box, hands
- 1 - DS3231 Real Time Clock Module
- 2 - 5V-6V power supply
- 2 - Male to female power supply adapter
- 1 - Infrared receiver + remote sender
- Wood, angle brackets, nails, superglue, etc.









Hardware

Each NEMA17 motor is a 4-pin stepper motor connected via wires to a L298 driver

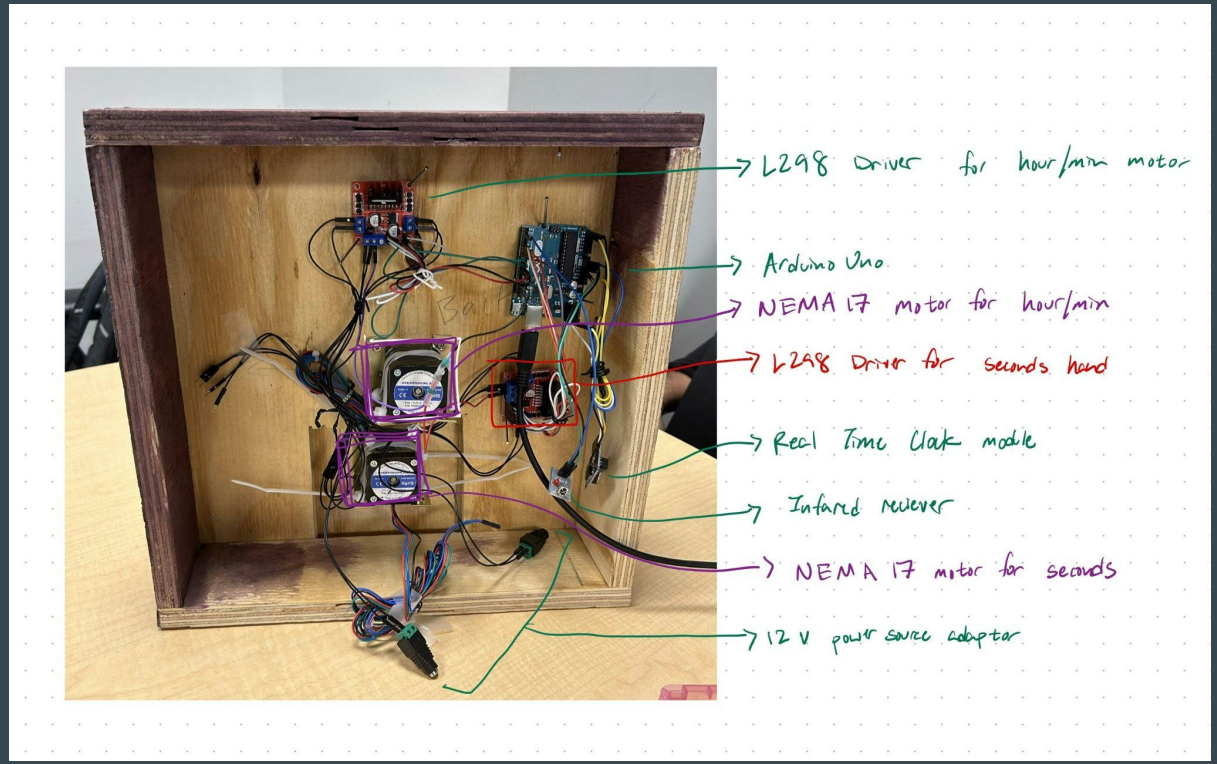
Each driver has connections to its respective motor, 4 control pins on the Arduino, a power supply, and two grounds - one to the power supply and one to the Arduino.

The individual 5V-6V power supplies are used for the motors, wired through a male-female adapter.

A custom wire was created by our team to split a single ground on the Arduino to two motor drivers.

The NEMA motors were mounted to the wood with metal brackets and fastened with zip ties to prevent excessive vibrations or movement.

Wires and cables were tied together to allow for better cable management, organization, and easier debugging process.



Software

- “AccelStepper” Library
  - Manipulates movement of motors with acceleration and deceleration
  - Position switching
  - Pausing, delays
  - Clockwise and counterclockwise rotation
- Helper functions to setup the RTC, IR remote, and assist in other functions
- Setup()
  - Set up max speed, acceleration, initialize components
  - Initialize positions around the clock for both motors, accounting for the gear ratio of the gearbox
- Loop()
  - Mode switching
  - Control movement of the motors/hands

Challenges Encountered

- RaspberryPi versus Arduino support for NEMA17
  - We eventually switched to Arduino from RPi due to better library support for NEMA17 motors
- Choosing the proper library for high speed motor movement
  - We settled on the AccelStepper library compared to offerings from Moba and the default Arduino stepper control library Stepper.h
- Limited dynamic memory for Arduino
  - We had to adjust our program extensively to use the Arduino's very limited memory with greater optimization
- Calibrating software commands with a complicated gear ratio
  - The motor was not able to step in non-integer numbers, so a complicated mathematical formula had to be developed for the stepping of the motor to prevent losing steps
- Driver overheating
  - The L298 driver is an inefficient driver and would often throttle and lead to unpredictable behavior of our motors - to prevent this, we programmed the motor's 4 pins to be set to LOW during the interval between each movement of the hands
- Switching modes leading to confusion for the motor's positioning
  - The NEMA17 motor is unaware of its position, so we had to create special solutions for the program to keep track of where it was to always present the accurate time
- Motor and gearbox connection was not secure
  - We initially utilized superglue to fasten these components together but after much trial and error, we opted to add malleable stainless steel binding to effectively 'strap in' the gearbox to the wooden case and create a secure fit.
- IR Receiver Malfunctioning

Improvement Areas

- Add EEPROM memory - Arduino library can hold values at different memory addresses so our clock can move to current time even after being unplugged.
- Speed timer mode: turn the hour/minute hands into minute/seconds, then turn the seconds dial into a milliseconds dial. Not very hard to implement.
- Improve transitions between states
    - Smoother and faster
- User set time (software)
- Better drivers - L298 had overheating problems, we had to implement solutions in the code (sending low signals when not in use).