Nixie Tube Clock Design

Ryan Michael Kissinger, BSME

ABSTRACT
This paper proposes a methodology to design, program, and manufacture a working Nixie Tube alarm clock, comprehensively describing the design process, necessary parts, and assembly methods used to develop the device.

READ THE DISCLAIMER IN APPENDIX F BEFORE PROCEEDING.

0. Introduction

After a long hiatus due to the efficiency of high-powered LED displays, Nixie Tubes have reemerged in public eye as a popular device for tinkerers, engineers, and hobbyists. This revolution is wrought not only by their historical significance to early computers, but also by their eye-catching neon aesthetic-- their glass bulbs housing radiant filaments of numbers, decimals, letters, and characters, their planes of twisted wires illuminating with a luminous neon glow

The history of Nixie Tubes dates back to the 1950s and 1960s, an era long before the existence of low-powered LEDs, employed in industrial and scientific automated equipment. After its conception by a German-American hobbyist, and later commercialized for public use, the Nixie Tube has played pivotal roles in American technological history, from moon landings to Wall Street [1].

Through the hindsight of our privilege by modern technology, it can be easy to regard the Nixie Tube as an overcomplicated solution to a simplistic problem. But before low-power, efficient, light-emitting-diode displays came into significance, the Nixie Tube was a substantial breakthrough at the time. The Numerical Indicator Experimental-1 tube, shortened to the nickname "Nixie" for understandable reasons, contains a set of diodes in a glass tube containing neon gas. The numerals are cathodes aligned in parallel planes, and when a significant voltage difference is observed, the surrounding neon gas is ionized, affording Nixie tubes their infamous glowing effect [2].

To engineers, hobbyists, electricians, and artists, the Nixie Tube's importance has been reborn into culture through the pleasure of its aesthetic. In modern pop culture, Nixie devices have emerged in television and movies. My personal inspiration for this device arises from the Japanese time travel anime *Steins;Gate,* in which the main character Okabe Rintarou employs the use of a 8-bulb, IN-12 Nixie device called a "Divergence Meter" to numerically identify the reality that he is currently on. This project is the amalgamation of my passions for mechanical, electrical, and computer science engineering; a love letter to the major that has captivated me since I first began my educational journey.

This paper will systematically cover my procedure of designing a Nixie tube clock, exploring the following:

1. Initial considerations/parts selection;
2. Manufacturing drawings;
3. Circuit schematics;
4. Programming code;
5. Assembly and design;
6. Additional features.

1. Initial Considerations and Bill of Materials

Mechanical Components			
Component	Description	Qty	Example Source
M2.5 Standoff Set, 150-pc	6mm, 10mm, 15mm, 20mm size; Male-Female and Male-Male.	1	Amazon (Sutemribor)
Craft Wood Beam*	¼“ x 3” x 24” panels.	1	Online or craft store
Craft Wood Plane*	Minimum area 5.5” x 2.5”.	1	Online or craft store
Wood Glue	-	1	Online or craft store
Electrical Components**			
Component	Description	Qty	Example Source
SN74HC595D	Bit-Shift Register, Surface-Mount.	3	
K155ID1	Also known as К155ИД1 or SN74155N: Binary-to-BCD driver.	6	
ESP-32 HUZZAH	Adafruit ESP32 breakout.	1	Adafruit
2x8 16-Pin Header	Pin header for breakout.	1	
1x3 3-Pin Header	Pin header for breakout. Purchase in bulk by buying a pack.	Pack	Amazon, Mouser, DigiKey
1x6 6-Pin Header	Pin header for breakout. Purchase in bulk by buying a pack.	Pack	Amazon, Mouser, DigiKey
R0805 50 Ω	50 Ω resistor.	3	Mouser, DigiKey
35211220KFT 20K Ω	20K Ω resistor for high-voltage power.	6	Mouser, DigiKey
L7805CV	12V to 5V step-down converter.	1	Mouser, DigiKey
C0805 0.1µF	0.1µF 50V decoupling capacitor.	5	Mouser, DigiKey
C0805 10µF	10µF 50V decoupling capacitor.	3	Mouser, DigiKey
2.56mm x 2 Screw Terminal	Screw terminal for high-voltage wire connection from step-up converter.	1	Mouser, DigiKey
Female DC Plug	12V DC plug adaptor.	1	Amazon (UXCELL)
DC Power Cable	12V wall-plug power cable.	1	Amazon
NCH6100HV DC Step-Up Converter	12V to 200V step-up converter.	1	Amazon
6PCS IN-14 Nixie Tubes	Pack of 6 Nixie tubes.	1	Amazon

* Displayed in inches for convenience, since standard sizes are often sold with imperial dimensions. This project uses the SI metric standard.
** Make sure to purchase a wealth of extra components, as small components such as thick-film resistors and capacitors are incredibly easy to lose.

Table 1.1 | Bill of Materials for creating a Nixie tube clock.

The following table covers the components necessary to make a single Nixie Tube clock. Further description will highlight the purpose behind the selection of all specialized components.

1.1. SN74HC595 Bit-Shift Register

The **SN74HC595** is an 8-bit, serial-in, parallel shift-out register that holds the 4-bit binary values representing the digit on the Nixie tube. This device is capable of transmitting a numerical input over a serial connection into a physical manifestation of the represented number, opening and closing eight gates that directly represent the bits of the binary number. This device will be employed in conjunction with the **K155ID1**.

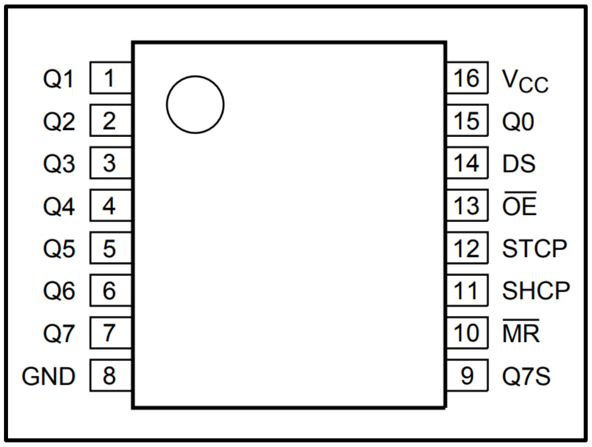


Figure 1.1 | Pinouts for the SN74HC595, adapted from Nexperia datasheet [3].

Pin	Description
Vcc	Power pin
GND	Ground
DS / Serial	Serial input.
$\overline{OE}$	Output enable (active when LOW)
STCP / RCLK	Storage register clock pin
SHCP / SRCLK	Shift register clock pin
MR	Master reset (active when LOW)
$\overline{SRCLR}$	Shift register clear
Q_A- Q_H	Bits 1-8, LSB to MSB
Q_H’	Daisy chain out (to next Serial connec.)

Table 1.2 | Pinouts for the SN74HC595.

What's important to observe is that the bits transferred into the shift register through Serial first enter the Q_A register, then are procedurally shifted towards the Q_H. Therefore, when we interface with this device, the LSB will end up being the last shifted value, housed in Q_A, and conversely, the MSB will house the first shifted value (in Q_H). Any further bit shifts will result in the Q_H bit being shifted to the next shift register, into its Q_A space.

Our end goal is to drive the nixie tubes. The **K155ID5** chip, which will be discussed next, is the component that allows the 4-bit binary number to be converted into a signal that drives one number at a time. By daisy-chaining three of the **SN74HC595** chips together, we have six available 4-bit binary numbers to drive our six nixie tubes. We use the Serial, RCLK and SRCLK to drive this process.

The **STCP** pin is the storage register clock pin. When it is low, the device "starts listening" for binary values to be introduced through the Serial line. When we change it to high, the device stops listening, then shifts the input data to the output line.

The **SHCP** pin is the shift register clock pin. The most important characteristic of this pin is its behavior on a rising edge transition – when it changes from low to high, all the values in our shift register are shifted by one place. Q_A is shifted to Q_B, Q_B to Q_C, and so on. The value in Q_H is shifted through the Q_H' pin, which connects to the next register's Serial line, entering it into its Q_A position.

The **Serial** pin holds the data transferred from the microcontroller: a single bit, zero or one, low or high. The value in this register is shifted in when the SHCP pin undergoes a rising edge transition.

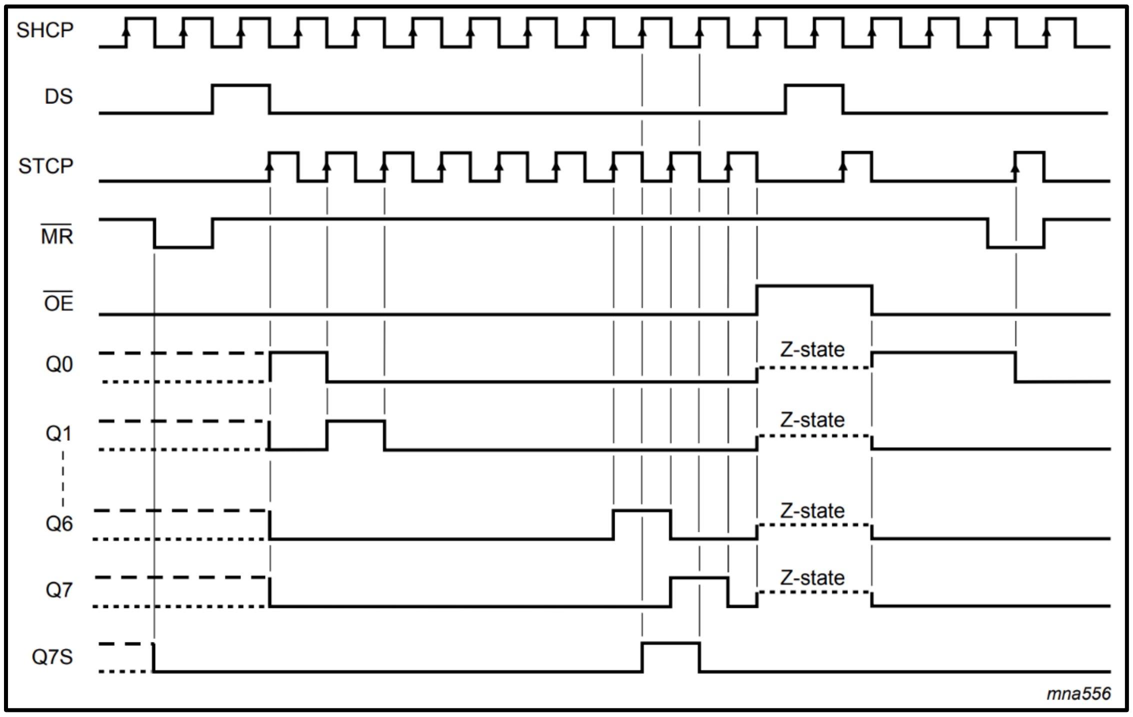


Figure 1.2 | Timing diagram for the SN74HC595, adapted from the Nexperia datasheet [3].

We observe this interaction in the above timing diagram table, adapted from a Nexperia datasheet [3]. The **SHCP** and **STCP** line share a relationship of one-half phase off from each other, as their behavior is contingent upon the rising-edge transition. When the **DS** (or Serial) pin receives a high signal, the **SHCP** pin receives a rising-edge signal from low to high, and the **SHCP** pin is low, the device processes the signal from the **DS** line and places it in the first register (Q_A). When the **STCP** line transitions to high, it transitions the input data to the output line and "stops listening" for new values.

The interplay between the **STCP** and **SHCP** line continues; each rising-edge transition of the **SHCP** pin drives the original signal from the **DS** pin one step further along the registers. Eventually, this signal reaches the Q7s or Q_H' line, and if the Q_H' line is daisy-chained to the next SN74HC595 **DS** line, that value is placed in its first register position.

1.2. K155ID1 Binary-to-BCD Chip

The **K155ID1** (also known as the К155ИД4 or the SN74155N) is a binary-to-BCD driver that converts our 4-bit nybble[1] of binary numbers into a signal for the correct digit on our Nixie tubes.

We observe on the pinout schematic that the values 0-9 are represented. For this particular chip (that only represents values 0-9), any value greater than 9 (or 0x0110, binary ten) results in no output. For convenience, we can use these values as "dead outputs" when we don't want to display a number[3].

A specialized feature about this chip is that it is directly intended for use with high-voltage Nixie tubes (after all, the documentation contains a Russian document from the cold war era). The Logic Diagram shows us that Zener diodes and PNP BJT logic is present in the device. Since BJTs are present, the signal obtained from our microcontroller is segregated from the high-voltage connection to the Nixie tubes running in excess of 200 volts. The segregation of these two voltage lines ideally prohibits corruption of the STM32 microcontroller, which runs on a standard input of 5 volts, which is exactly the same as the voltage provided over a Serial line.

Figure 2.3 displays the pinouts of the K155ID1 chip for quick reference. The pins marked A, B, C, and D represent the four input bytes. According to the logic of Table 2.3, the 4-bit input corresponds to the numerical outputs displayed. These ten output lines, numbered 0 through 9, are directly connected to the cathode wires of the Nixie Tube. The **Vcc** line allows the bipolar junction transistors in the integrated circuit to "switch" between inputs.

Binary	Input D	Input C	Input B	Input A	Output
0x0000	L	L	L	L	0
0x0001	L	L	L	H	1
0x0010	L	L	H	L	2
0x0011	L	L	H	H	3
0x0100	L	H	L	L	4
0x0101	L	H	L	H	5
0x0110	L	H	H	L	6
0x0111	L	H	H	H	7
0x1000	H	L	L	L	8
0x1001	H	L	L	H	9
(Over Range)[3]					
0x1010	H	L	H	L	-
0x1011	H	L	H	H	-
0x1100	H	H	L	L	-
0x1101	H	H	L	H	-
0x1110	H	H	H	L	-
0x1111	H	H	H	H	-

Table 1.3 | Binary input vs. output for the K155ID1 chip, adapted from the National Semiconductor Corp. datasheet [4].

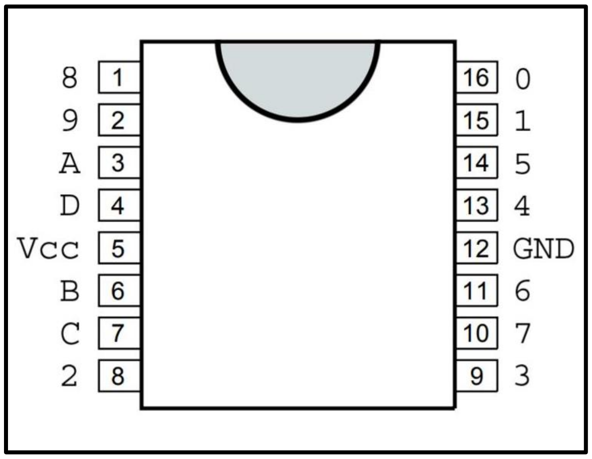


Figure 1.3 | Pinouts of the K155ID1 chip.

[1] A "nybble" is four bits, or half a byte.

1.3. Using Both ICs Together

Properly interfacing with the Nixie tubes requires us to understand how the two ICs communicate.

Although further programming description will take place in Section 4, Figure 2.4 illustrates the procedure used to transition binary values into numbers on the clock:

1. The numerical time is calculated through use of the C++ program.
2. Each ones- and tens-place digit is separated and stored into individual values, and then converted to binary.
3. Binary pairs are concatenated into 8-bit bytes for transference to the SN74HC595 chip using the shift operator.
4. The bit-shift operation takes place, shifting all three bytes into place in the SN74HC595 chip.
5. The information in the most significant nybble and least significant nybble are transferred to the K155ID1 chip. Each of these contain a 4-bit binary number representing the digit to display on the Nixie tube.
6. The high voltage signal through the anode of the Nixie tube is open on all cathode lines, except for the desired number, causing the requested number to glow.

Further description of the wiring diagram will be provided in Section 3 (Circuit Schematics), and program code for interfacing with both of these devices will be provided in Section 4.

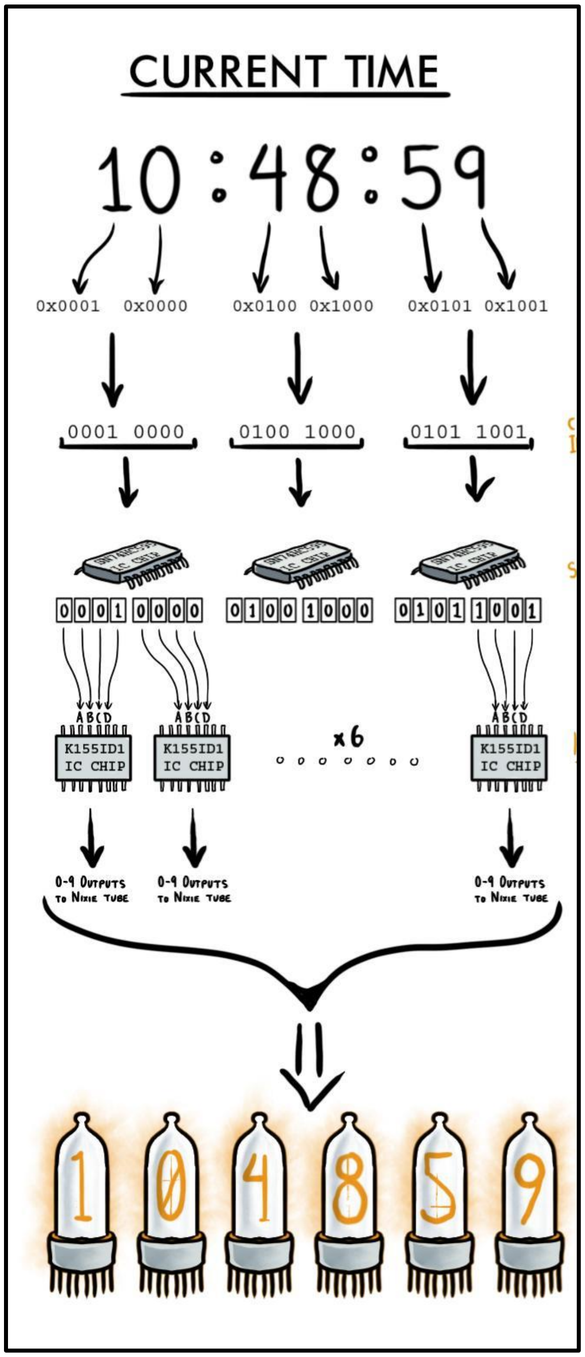


Figure 1.4 | Representation of the procedure to extract digits from time and use both ICs to transmit them as signals to the Nixie tubes.

1.4. ESP32 Adafruit Featherboard

The Adafruit ESP-32 Huzzah microcontroller is an excellent featherboard for prototyping purposes. To save time constructing the Nixie tube clock, using a featherboard prevents us from soldering thermal ground planes and using a heat gun to solder intricate components. Take, for example, the USB-to-UART chip already present on the ESP32 feather — with terminals nearly as small as the width of a human hair, this process requires specialized equipment beyond the budget of a simple hobbyist. In an effort save time, effort, and frustration, the featherboard is employed in this design. In addition to this, its power equates well with the requirements of the project: with four distinct timer channels, a sufficient number of additional GPIO pins, and low power requirements and power consumption, this makes the ESP32 an excellent choice [5].

Using Sloeber, an IDE built off of the Arduino workspace that packs a lot more power, we can interface with the ESP-32 quickly to prototype code. The procedure required for installing Sloeber for the ESP32 can be found in Appendix A, on Page 31.

1.5. NCH6100HV Voltage Step-Up

The NCH6100HV, designed by Yan Zeyuan, supplies us with the voltage required to drive the Nixie tubes. This step-up converter accepts a 12V signal and amplifies it to 200V, supplying the filament with enough power to ionize the neon gas.

An important calibration is required to accurately step the voltage up to 200 volts. On the circuit board there is a potentiometer that increases the high voltage output with respect to the clockwise rotation angle. From the starting orientation shown in Figure 1.5, my module required at least a 270-degree rotation to break 200 volts.

The test shown in Figure 1.6 allows us to determine the correct calibration, as some devices may differ depending on the voltage. For a standard 12V input, specified by the Bill Of Materials in Table 1.1, we can simulate this voltage by using a DC power supply and setting up a simple circuit to test this. Both

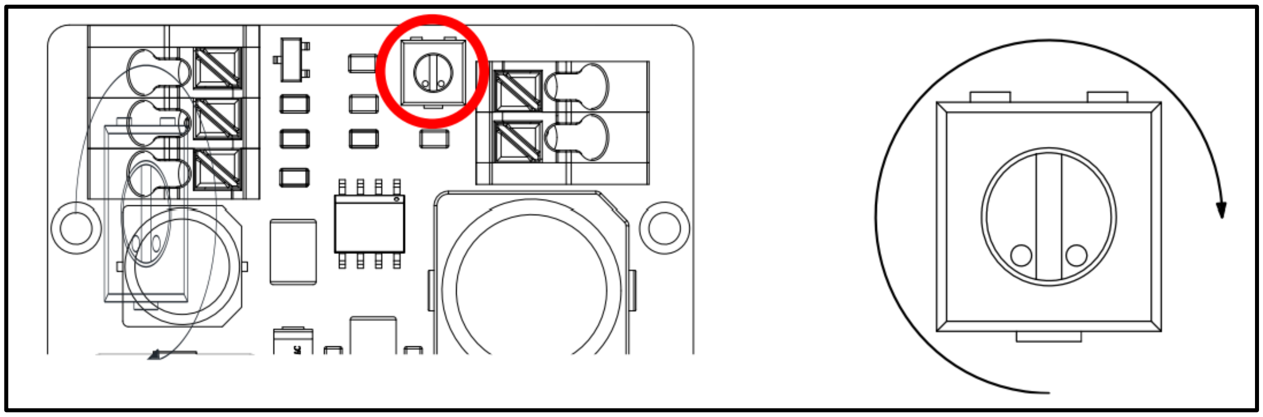


Figure 1.5 | Voltage step up potentiometer, adapted from the NCH6100HV datasheet [7].

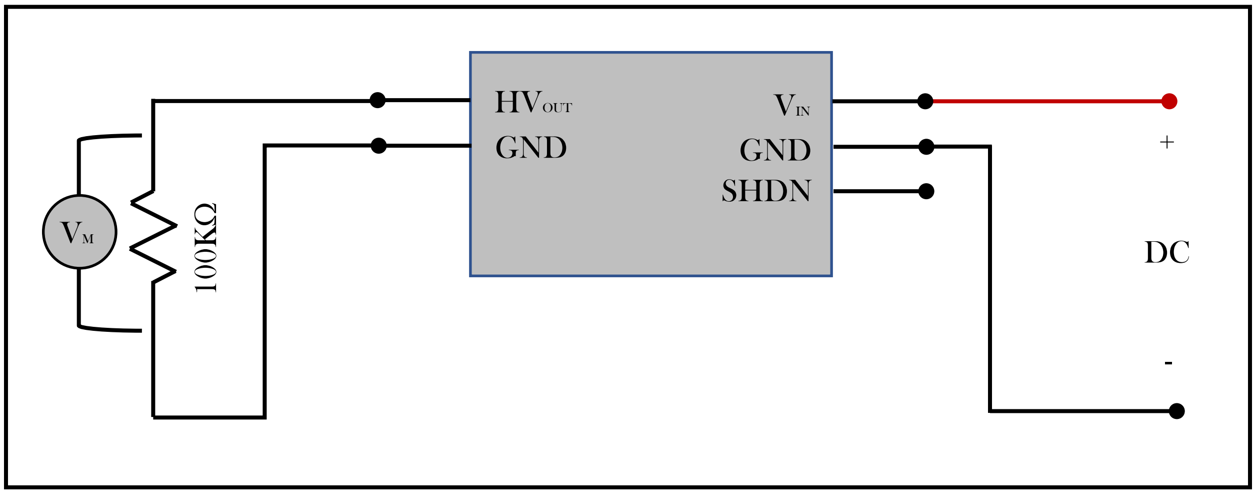


Figure 1.6 | Standard NCH6100HV voltage test setup.

GND terminals of the NCH6100HV are connected to their respective ground and voltage lines; the input to the positive and negative leads of the DC power supply, and the high voltage ground-and-power to the rest of the circuit.

Iteratively turn the potentiometer half a quarter-turn at a time. Warning— For safety purposes, each time ensure total deactivation of the input voltage line: turn off the DC power supply and disconnect both leads before adjusting the potentiometer with a screwdriver. Read and understand the disclaimer in Appendix F before proceeding, and disconnect all voltage sources before handling electrical devices.

The SHDN line, shown in Figure 1.6, can also be used to turn on or off the output of the NCH6100HV. A high signal disables the high voltage line, while a low signal enables it. This can be useful when connected to a GPIO pin on the ESP32 module, especially since Nixie tubes have a minimum reported life of approximately 5000 hours. Coupling the device with a motion detector could activate the clock only when a viewer is present, saving power and ensuring longer life of the tubes.

For further information on efficiency curves and power specifications, reference the datasheet.

2. Manufacturing

2.1. Engineering Drawings

Since most Nixie tube clock-builders don't have access to a machine shop, I've simplified the design process to accommodate these limitations. Besides using standoffs for the printed circuit boards, no fasteners are required; the whole manufacturing process can be accomplished with wood glue, a drill, and an XActo™ mini-saw (or equivalent).

Appendix B1 displays the Bill of Materials with further description of the components used in the design. Figure 2.1 shows the exploded view of the Nixie tube clock, where the ③**top PCB** and ②**bottom PCB** are separated by standoffs. The bottom PCB has the ④**ESP32 module** attached as a hat, or soldered; whichever is the easier configuration. The ⑥**NCH6100HV** is mounted to the bottom of the ①**enclosure box** by either adhesive or two M1 screws.

Four different standoffs are used in this design. The balloons in Figure 2.2 and Figure 2.3 illustrate this:

- ⑦ **M2.5x20mm + 6mm Thread**
- ⑧ **M2.5x6mm**
- ⑨ **M2.5x15mm + 6mm Thread**
- ⑩ **M2.5x6mm + 6mm Thread**

The top of the assembly contains the ③**top PCB:** soldered to this are the three ⑬**yellow LEDs** and the ⑪**IN-14 Nixie tubes.** The ⑫**mounting cover** fits directly over all of these components and is bounded by the topmost standoff.

Figure 2.4 displays the front view detail drawing from Appendix B2. The broken-out

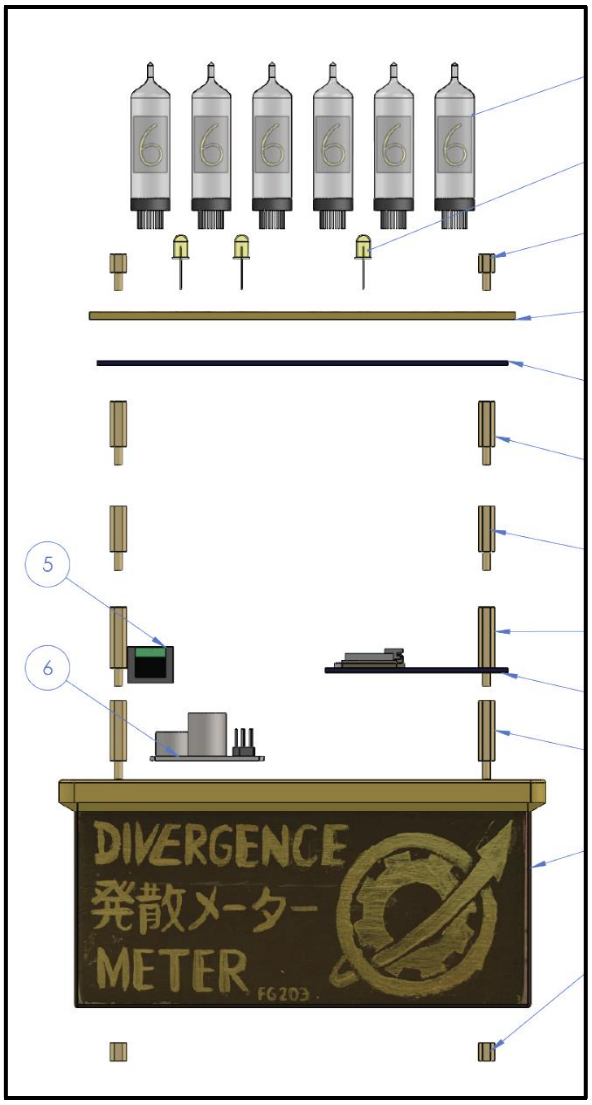


Figure 2.1 | Exploded view of Nixie tube clock assembly.

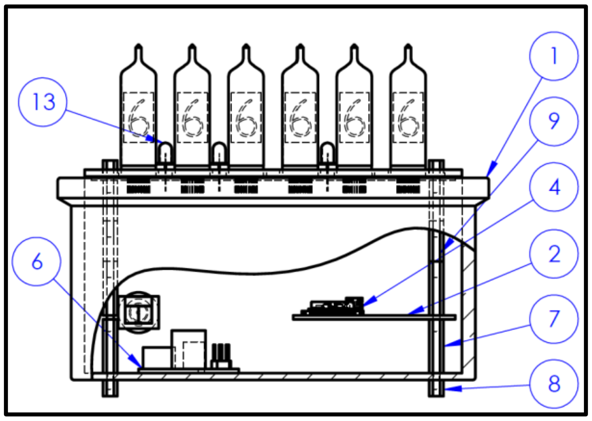


Figure 2.2 | Side view of full Nixie clock.

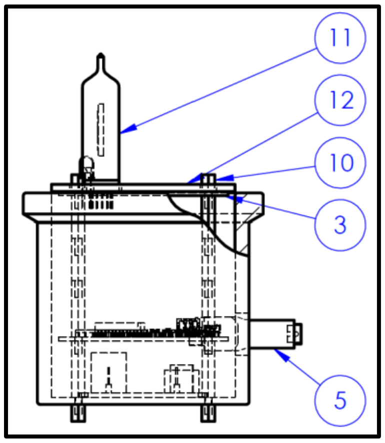


Figure 2.3 | Front view of full Nixie clock.

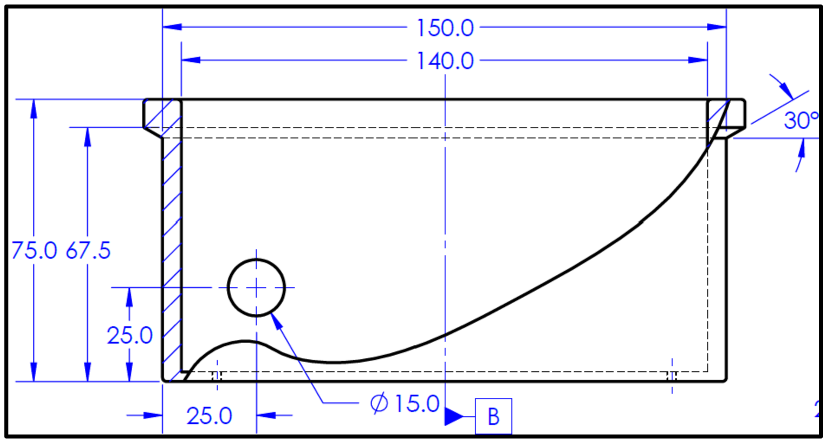


Figure 2.4 | Front view detail drawing of ①**enclosure box.**

section displays the diameter hole that accepts the ⑤**12V adapter,** which is press fit from the inside then fixed in place with adhesive.

The box can be constructed using four wooden walls, a bottom, and the top cover, but the design I've illustrated adds a slight ∠30° chamfered edge. This optional aesthetic is achieved using a small square rod of approximately 8.0mm square cut into four sections to form a perimeter around the box. After wood glue adheres this feature to the box, the chamfer can be added through sanding.

Appendices B3-B6 have 1:1 drawings that can be printed out for the cutting and drilling process. B3-B5 contain the walls of the box, while B6 contains the top mounting plate. Positional, maximum material condition GD&T callouts require the position of the holes to accommodate the ⑬**yellow LEDs**, the ⑪**IN-14 Nixie tubes,** and the ⑫**mounting cover.** The mounting cover also provides a safety feature, preventing electrocution from high-voltage, exposed wires running from the HV power supply to the Nixie anodes. In order to accommodate the profile of the circuit board, the bottom

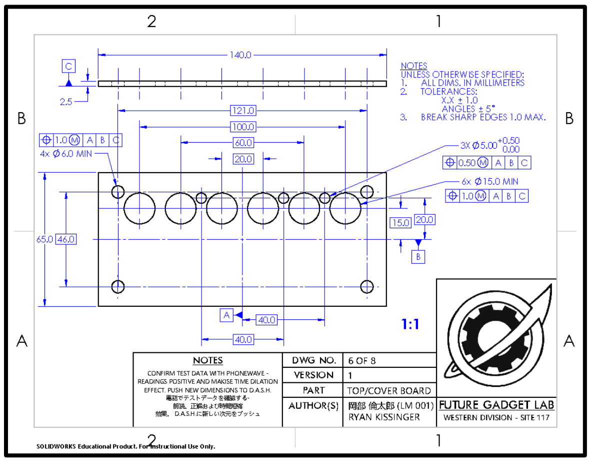

140.0
121.0
100.0
60.0
20.0
4x ∅6.0 MIN
3X ∅5.00 +0.50 0.00
6x ∅15.0 MIN
65.0
46.0
15.0
20.0
40.0
40.0
1:1
NOTES
UNLESS OTHERWISE SPECIFIED:
1. ALL DIMS. IN MILLIMETERS
2. TOLERANCES: X.X ± 1.0 ANGLES ± 5°
3. BREAK SHARP EDGES 1.0 MAX.
DWG NO. 6 OF 8
VERSION 1
PART TOP/COVER BOARD
AUTHOR(S) 岡部 倫太郎 (LM 001) RYAN KISSINGER
FUTURE GADGET LAB
WESTERN DIVISION - SITE 117
SOLIDWORKS Educational Product. For Instructional Use Only.

Figure 2.5 | Top cover from Appendix B6.

plane of the mounting cover can be sanded to-shape.

Figure 2.5 previews the sheet attached in Appendix B6, along with all the positional tolerances required for assembly. The following positional tolerances are observed.

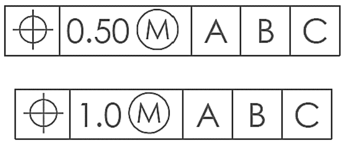


Figure 2.6 | GD&T positional tolerances.

Although the tools recommended for manufacturing the box afford little accuracy, the alignment of these components is detrimental to ensuring proper fit of the top board. This GD&T callout ensures the hole is sized large enough to allow the Nixie tubes and LEDs are able to fit through, while not drastically exceeding ideal dimensions. The spacing between the LED holes and the Nixie tube holes is tight, so proper positioning tolerances ensure no manufacturing errors produce merged holes.

2.2. Manufacturing

Use the 1:1 drawings in Appendices B3-B6 as a reference to cut the craft wood. Print two copies of the BOARD, LT+RT, as the left and right sides of the Nixie clock are the exact same. Note that the drawings provide the board width as 5.0mm; if a different board width is desired, adjust the measurements accordingly. Also print two copies of the TOP BOARD, as the bottom board holds the same perimeter dimensions. Drill holes according to the positional tolerances in Appendix drawing B2.

The drawing in Appendix B2 displays a top view of the box. Regardless of the width of the wood selected for the outer frame, the internal dimensions of the inner opening will remain consistent. B2 also displays the chamfered upper edge, which is an optional feature (as previously discussed.)

Nails are an acceptable option to fix all sides of the box. In an effort to make this project require minimum tools and simplify the assembly process, this is also achievable with wood glue. After ensuring the sides of the box hold proper perpendicularity, lather wood glue between surfaces on the corner joints.

Once all surfaces have properly been coated and oriented, stand the frame upright, and allow proper time for drying. Wood glue often dries stronger than the wood itself. During the drying process, I created internal fillets by applying glue along the 90° edges to ensure exceptional strength.

With the outer bounding walls of the box constructed, adhere the bottom board to the box. After the assembly has dried, screw standoffs into the M2.5 holes on the bottom board.

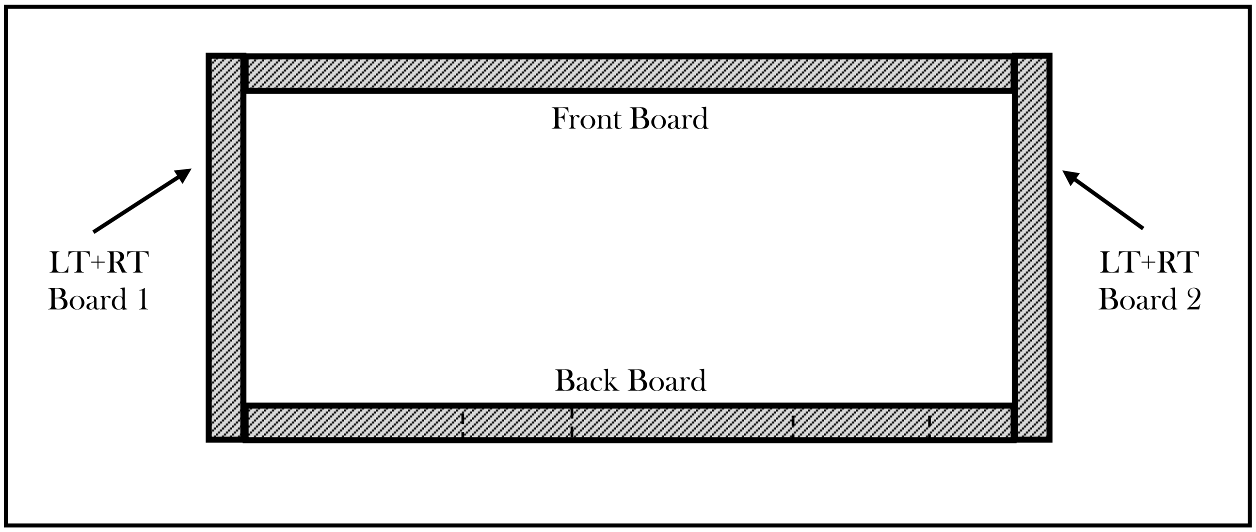


Figure 2.7 | Proper orientation of the boards forming the enclosure box.

The outer chamfer design adds aesthetic to the box, but is completely optional. To achieve this, I used four pieces of an 8.0mm square rod, glued them to the upper part of the box, and sanded them down to shape.

Sand the faces of the box to correct any imperfections from the corner joints, ensuring flat planes on all four sides. Once the entire assembly has been completed, I recommend painting the box before sanding the edges. I used two different paint colors: dark brown to maintain a classic aesthetic, and gold to embellish the letters on the front of the box. Once the box has been painted, sanding the edges adds a touch of a "worn" aesthetic. This feature can easily be corrected by re-painting.

After soldering all the boards, fix the NCH6100HV high voltage power supply using two M2.5 screws. The position is shown in figure 2.2 by Balloon 6.

Attach the bottom (small and square) PCB in proper orientation to two standoffs. Insert the DC adapter into the back hole of the Nixie clock. Connect two male-female DuPont wires to both terminals of its screw terminal, and connect the remaining female side of each wire to the input/low voltage side on the NCH6100HV power supply.

Wire the HV side of the NCH6100HV power supply with two female-female DuPont wires[2]. These will later be connected to the JP2 pin header on the top board, to ports 1 and 2 (shown in Appendix C1.)

Connect female-female DuPont connectors to all seven pinouts on the bottom board (shown in Appendix C3.) With the top board not yet connected to all four standoffs, connect the wires from the bottom board and the NCH6100HV high-voltage output to the underside pin headers:

- GND to GND.
- HV to +180-220V.

Connect the bottom board DuPont wires to the top board as follows:

- One +5V bottom pinout to the top +5V line.
- PULSE to PULSE.
- SHCP to CLOCK.
- STCP to LATCH.
- DS to DATA.
- DIVLED to DIVLED.

Space the bottom board from the top board using standoffs. For the other two holes that were unused by the bottom board, screw these two standoffs to the same height as the other two.

By this point, all four pillars of standoffs should reach approximately the height of the enclosure's top surface. Attach the top cover to the top of the upper PCB, then fasten screws/small standoffs to join all four M2.5 holes, sandwiching the top cover and upper PCB board together[3].

Once all connections have been completed, connect the 12VDC adapter to the wall supply cord. The device should properly boot and display the time.

[2] If control of the high voltage connection is desired, connect a third female-female DuPont wire to the SHDN terminal shown in Figure 1.6. Connect this to one of the unused GPIO pins discussed in section 3.6.

[3] If needed, the top cover and top PCB can be separated with standoffs instead.

3. Circuit Schematics

3.1. Power Considerations

Since the device uses two distinct voltage lines, it's critical to use proper circuit design to segregate them. Powering the Nixie tubes requires nearly 200 volts. Without using a device embedded with BJTs or Zener diodes to separate these voltage lines, electrical failure is almost certainly assured.

The other line operates on a standard 5V supply. This powers the ESP32 module, LEDs, and logic-operating components (which are often stepped down to 3.3V through the ESP32 feather board.)

In order to power both these supply lines while using a single voltage source, we use a 7805TV voltage regulator. Since our device obtains power from a 12V wall power source, we can use this source to power both the 200V supply line and the 5V supply line, while maintaining a proper degree of separation.

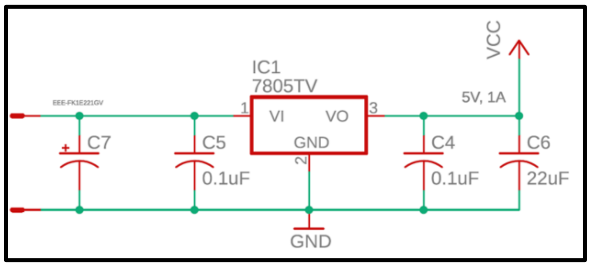


Figure 3.1 | Voltage regulator circuit component. See Appendix C3 for complete drawing.

In Figure 3.1, the circuit schematic for the voltage regulator is displayed. The leftmost side is supplied with 12VDC from a wall source. On each side, two decoupling capacitors filter out noise: the left side with 0.1μF and 10μF capacitances; and the right side with 0.1μF and 22μF capacitances. This ensures any additional noise during operation of the nixie clock is filtered out and supplies our microcontroller with the standard 5V required.[4]

The 12V supply line is also connected to the NCH6100HV high-voltage boost converter, which supplies 200V to the Nixie tube circuit.

3.2. PCB Separation

As shown in Appendix C2 and C3, the circuit for the Nixie clock has been divided into two PCB boards: an upper board to mount the Nixie tubes, LEDs, and logic; and a bottom board to house the ESP32 and 7805TV voltage regulator.

Both PCBs have sets of pin headers.

Lower Board – 1x06 JP2

- **CLOCK**: Clock/SHCP pin.
- **LATCH**: Latch/STCP pin.
- **DATA**: Serial/DS pin.
- **PULSE**: Drives both time LEDs.
- **DIVLED:** Powers first LED.
- **PIN21:** Optional GPIO pin.

Lower Board – 1x03 JP1

- **5V:** Two additional pins breaking out the 5V supplied by the voltage regulator.
- **GND:** Ground pin.

Lower Board – 2x08 JP3

- Used to break out the rest of the components available to the ESP32 module.

Upper Board - 2x06 JP1

- Connects pins to the SN74HC595 surface mount chip.

[4] Ensure the USB Type A and 12VDC power supply are NEVER connected at the same time. This can lead to serious issues with the microcontroller.

Upper Board – 2x03 JP2

- **+5V:** Connected to lower board 5V supply pin.
- **HV**: Connected to HV side of the NCH6100HV high-voltage power booster.
- **GND:** Common ground, connected to the lower board GND.

Both boards' M2.5 holes are concentric. When looking down on the Nixie clock from the top, with the front pointing away, the bottom board is mounted to the right two standoffs.

3.3. IC Connections

Figure 3.2 displays both the high voltage and logical voltage supply lines, separated by the SN74141 chip.

The 74HC595 has 16 total pins. Q0 through Q7 are the logical pins representing the 8-bit register, while Q7s (or Q7') represents the daisy-chain output pin, connected to the next DS/Serial pin.[5]

The $\overline{\text{MR}}$ pin and VCC are connected to +5V. The Master Reset pin is set high to disable reset of the registers. GND and Output Enable ($\overline{\text{OE}}$) are grounded to enable output. DS, STCP, and SHCP are connected to the ESP32 microcontroller, driving the logic of the whole process. STCP and SHCP are wired to every SN74HC595 shift register, as the logic of the process requires every register shifting at the same time; without all registers connected, the Q7s would be unable to shift binary values to the next register.

Q0 through Q7 are separated into four-byte packages that are each wired to a K155ID1 chip. The logic of bytes A, B, C, and D determines the active numerical gate on the

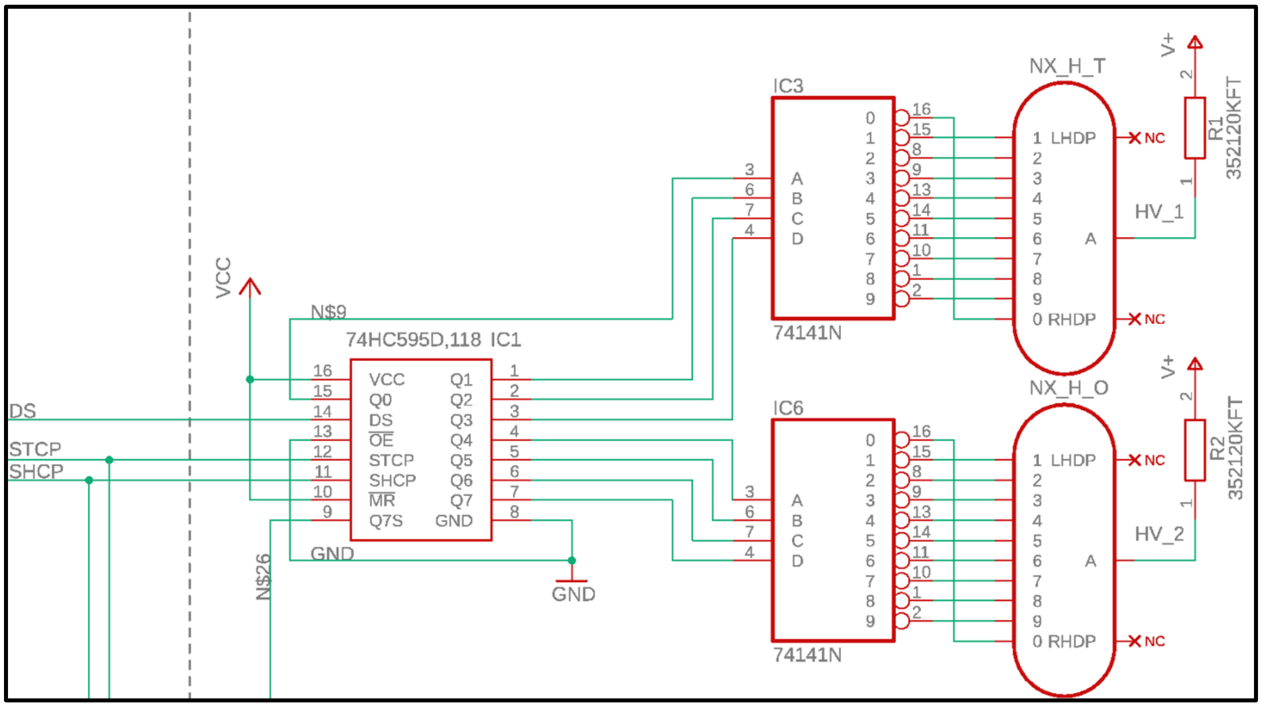


Figure 3.2 | Integrated circuit connections. Shows both the +200V high voltage side and the +5V logic side, separated by the SN74141 chip. See Appendix C1 for the full upper board schematic.

[5] See Figure 1.1 and Table 1.2.

right-hand side of the chip through use of BJT and Zener diode logic, separating the high-voltage line from the logical line (although all voltage lines share a common ground.)

The Nixie tubes are wired to all 0-9 terminals of the K155ID1 Binary-to-BCD chip. Left-hand decimal point and right-hand decimal point wires are left disconnected since they are not in use. The "A" terminal accepts the high-voltage line. Dividing the Nixie tube and the HV line is a 120KΩ resistor, to ensure proper current and correct voltage drop between the Nixie tube and ground.

Please reference Appendix C1 for the full upper board schematic to fully understand how the Q7s daisy chain connection works, as well as how the STCP and SHCP lines are connected. Beyond daisy-chaining the Q7s line to the next serial input, the other two 2-Nixie groups are wired exactly the same as shown in Figure 3.2.

3.4. LED Connections

Three LEDs are used in this design: two separating the hours, minutes, and seconds, and one separating the tens' and ones' place of the hours.

Recall the Nixie tube clock was originally designed to play homage to a device from Steins;Gate, the Divergence Meter. The device shows "world-lines," following the notation of a number between 0 and 10 with six significant figures. This LED is an optional addition that can be used to display custom messages at the top of the hour.

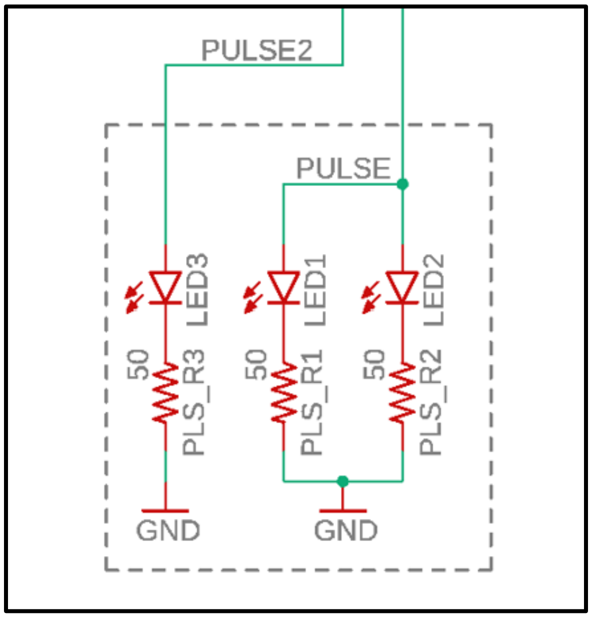


Figure 3.3 | LED connections for time separator line and first LED line. See Appendix C1 for the full schematic.

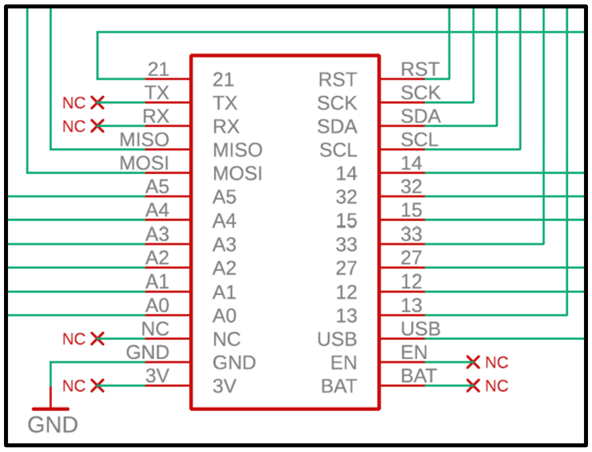


Figure 3.4 | ESP32 schematic for the bottom board. See Appendix C3 for the full schematic.

3.5. ESP32 Connections

Figure 3.4 shows the breakout for the ESP32 featherboard. The USB pin acts as the +5V supply for the microcontroller, and GND grounds the device to the project's common ground.

The rest of the GPIO pins are broken out for either connection to the upper board, or optional logic that can later be added. Pins 14 through 33 are used in this project. Pins used in this project are listed in the following table.

GPIO Pin	Description
21	Optional buzzer control*
14	Clock/SHCP
32	Latch/STCP
15	Data/Serial/DS
27	First LED control
12	Time separator LEDs

Table 3.1 | Pins used for the Nixie tube clock. *Connection used for an optional feature.

3.6. PCB Design Considerations

Appendices C2 and C3 display the EAGLE™ PCB renderings.

The top PCB has been designed for Nixie tubes and LEDs to be directly soldered.[6] The 74HC595 shift register, 50Ω, and 150KΩ resistors are the only surface-mount components; although the K155ID1 chip would ideally follow suit, no surface-mount packages are available (due to the antiquity of this component.) A better visualization of these spacings, without the air wires and mounted components, can be found in Appendix B6, as the top/cover board directly mates with the top circuit board's features.

The bottom PCB is mounted directly below the Top PCB. The two M2.5 holes are concentric with the two left holes of the top PCB, separated by standoffs. The ESP32 featherboard is soldered to the right side of the PCB[7] with the USB Type A female port facing downwards. Programming the ESP32 either requires removal of the PCB or drilling a hole into the side of the box so that a USB Type A cable can connect to it[2].

[6] Since Nixie tubes only have an estimated 5000 hours of life, using Mill-Max breakout socket pins can facilitate the replacement process.

[7] Instead of soldering the ESP32 directly to the board, a female pin header can be soldered here instead, as the hole distance follows the standard 2.56mm separation.

4. Program Code

Now that all the engineering schematics and electrical circuit designs are established, the program code can be fully understood. The language delegated for this project is the industry standard, C++; as a mid-level language (compared to high-level languages like Python or low-level languages like Assembly), we can easily interface with the ESP32 microcontroller. Personally, I prefer to use a program called Sloeber that extends the strength of the Arduino environment into higher-level programming for more demanding projects.

The state diagram shown below in Figure 4.1 demonstrates the logic for the finite state machine design. State 1 serves as a hub state at which time is continuously displayed. Changing the time state is contingent upon user interaction with left and right buttons (denoted as the ⮈ and ⮊ symbol, respectively). Pressing both buttons together designates an "accept" command, while left or right either increment or decrement the selected value. Upon an accepted double-press, the program accepts selection of whether hours, minutes, or seconds are to be changed, starting in the hours' selection by default. Once the double accept command is entered, this quantity can be decremented or incremented by the left and right buttons, respectively. When the quantity has been adjusted to the correct value, entering the accept command again progresses the program back to the hub/time state.

This feature, or course, is optional, but highly recommended: Section 4.2 details configuration of the start time at boot. Incorporating button control is advised to enhance user interface with the Nixie clock, as it significantly simplifies the process.

State 5 is also an optional state. For my own design, the numerical display flicks through random numbers before settling on the top-of-the-hour time. This state exemplifies how additional states (e.g. a motion sensor state to prevent overuse of the Nixie tubes when nobody is around) can be added to further modify the clock. This is also why our circuit schematic in Section 4 allows for additional pinouts.

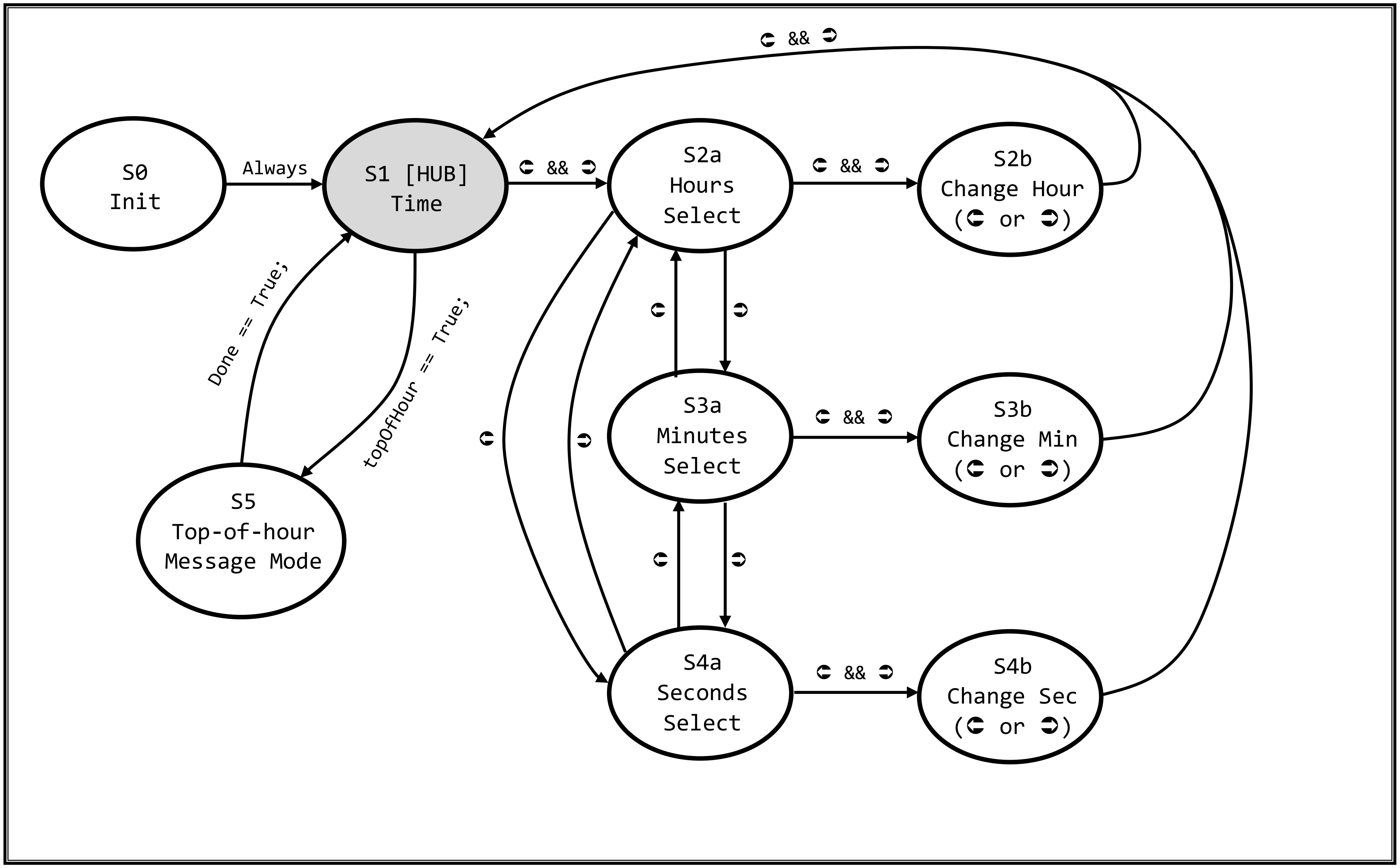


Figure 4.1 | State diagram for the program's finite state machine.

4.1. Table of Variables

The following table lists all variables in the main program. Variables with an asterisk are related to an optional feature that can be added on. See Section 5 for additional feature options.

Variable	Type	Description
Pinout Variables		
latchPin	Static Int	STCP on the 74HC595.
clockPin	Static Int	SHCP on the 74HC595.
dataPin	Static Int	DS/Serial on the 74HC595.
pulsePin	Static Int	Drives two top-mounted LEDs that act as separators for the hour.
buzzerPin*	Static Int	Drives the buzzer for sound.
incPin	Static Int	Button incrementing time.
decPin	Static Int	Button decrementing time.
statusPin	Static Int	Displays status of time selection.
divPin*	Static Int	Drives first LED separating hour tens and ones place.
timebuf	Static Int	-
Time Variables		
dataH	Byte	Stores hour value in binary form.
dataM	Byte	Stores minute value in binary form.
dataS	Byte	Stores second value in binary form.
currenttime	Int	Time extracted from the millis() function.
mil[2]	Double array	Records and stores previous and last time state for comparison.
d_mil	Double	Calculates time change from (mil[1] - mil[0]).
T_msec	Double	Time in milliseconds. Used for pulsing of LEDs and timing music notes.
pulseglow	Int	Holds the active value for the PWM cycle of the blinking time LEDs.
T_sec_act	Double	Active time in seconds.
T_secs	Double	Time in seconds. $\{0 \leq T\_secs \leq 59\}$
T_s_ones	Int	Seconds ones' place.
T_s_tens	Int	Seconds tens' place.
T_mins	Double	Time in minutes. $\{0 \leq T\_mins \leq 59\}$
T_m_ones	Int	Minutes ones' place.
T_m_tens	Int	Minutes tens' place.
T_hrs	Double	Time in hours. $\{0 \leq T\_hrs \leq 59\}$
T_h_ones	Int	Hours ones' place.

T_h_tens	Int	Hours tens' place.
T_days	Double	Counter for number of days passed.
Gmult*	Double	Timer that slows or accelerates BPM of song.
Song Variables*		
GOS_duration[13]*	Double array	Contains the duration of each note.
GOS_highnote[13]*	Double array	Contains the frequency of each note.
EstablishSongStart*	Bool	Returns if song has started.
SongEnd*	Bool	Returns if song has ended or not.
SongStart*	Double	Start time calculated from when first run.
SongTime*	Double	Song time relative to the start time.
NoteTime*	Double	Current value pulled from GOS_duration[i].
ActiveNote*	Int	Current note pulled from GOS_highnote[i].
Custom Message Variables "DivMode" *		
DivMode_SO[20]	Int array	Holds seconds ones' value for message.
DivMode_ST[20]	Int array	Holds seconds tens' value for message.
DivMode_MO[20]	Int array	Holds minutes ones' value for message.
DivMode_MT[20]	Int array	Holds minutes tens' value for message.
DivMode_HO[20]	Int array	Holds hours ones' value for message.
DivMode_HT[20]	Int array	Holds hours tens' value for message.
DivNumber	Int	Current displayed number. Fed into DivMode arrays to pull respective values.
Finite State Machine / Program Variables		
OpMode	Int	Operation mode.
BothPressed	Int	Returns duration both buttons pressed.
DecPressed	Int	Returns duration left button pressed.
IncPressed	Int	Returns duration right button pressed.
TimeMode	Int	Specifies time mode.
ChangeTime	Bool	Is the time to be changed?
DoublePress	Bool	Is there a double press?
DecPress	Bool	Is there a left button press?
IncPress	Bool	Is there a right button press?
PressButtonBuf	Double	Buffer preventing rapid button presses.
T_0	Int	-
SecondPassed	Bool	-
T_ms0	Double	-
MSPassed	Bool	-
Currentms	Double	-

Table 4.1 | Program variables.

4.2. Setup/Initialization

In the initialization loop, program variables are set to proper initial values. The following declaration establishes the second value that the clock starts at from boot:

```
//                      Hour        Minute    Second
  double T_sec_act = (11*3600) + (20*60)  +  (02);
```

Snippet 4.1 | Calculation of initialization starting time.

4.3. Time Control

This program operates based upon the innate Arduino function `millis()`, which returns the time that has passed since the board was powered on. We obtain the time difference (since the last time we checked) by comparing our new value with a previous value, then store it into the double d_mil to later add to a variable storing the active time in seconds, dividing this value by 1000 to obtain the exact current time in seconds.

```
  int currenttime = millis();
  mil[1] = mil[0];
  mil[0] = currenttime;
  double d_mil = mil[0] - mil[1];
  double T_sec_act += d_mil/1000;
```

Snippet 4.2 | Current time calculation from millis().

This time value, stored in the double T_sec_act, is used to calculate all the rest of the values. For sensitive time values that require the use of significant figures beyond whole numbers, the double type is used; otherwise, integer types store tens- and ones-place values for binary conversion.

```
PressButtonBuf += (d_mil/1000);
T_days = (int)(T_sec_act/3600)/24;

double T_hrs = T_sec_act/3600;
while (T_hrs > 24)
    T_hrs -= 24;
T_h_ones = (int)((int)T_hrs % 10);
T_h_tens = (int)(((int)T_hrs/10) % 10);

T_mins = T_sec_act/60;
while (T_mins > 60)
    T_mins -= 60;
T_m_ones = (int)((int)T_mins % 10) ;
T_m_tens = (int)(((int)T_mins/10) % 10);

T_secs = T_sec_act;
while (T_secs > 60)
    T_secs -= 60;
T_s_ones = (int)((int)T_secs % 10);
T_s_tens = (int)(((int)T_secs/10) % 10);
```

Snippet 4.3 | Deriving tens' and ones' values from current time.

Each ten-and-one time set for seconds, minutes, and hours first requires dividing out our T_sec_act value. From T_sec_act, we drive a double representation of the hours, minutes, and seconds. This is then converted to an integer type, and the Modulo (%) operator is used to extract both the tens and ones place. In order to keep our time values within the proper bounds (so that minutes or seconds, for example, don't surpass 60), we use a `while()` loop to constrain these values.

4.4. Defining a Proper Button Press

To prevent noise from affecting the accuracy of the buttons, we must clearly define what constitutes a button press for our program. Since our finite state machine passes through the `loop()` code many times a second, we check both of our buttons for any state changes:

```
bool dec = digitalRead(decPin);
bool inc = digitalRead(incPin);
```

Snippet 4.4 | Button press check.

Since we now have variables representing the state of each button, we assess how long they've been pressed. How many consecutive passes have they been pressed? After testing by printing button presses over serial, I found that a value around 10 is most effective for defining a button press. We use the integer type variables BothPressed, DecPressed, and IncPressed to count consecutive presses. If this cycle is broken, or the button is not held down for ten cycles, these variables reset themselves to zero.

```
// Defining double press
if ((dec == 1) && (inc == 1))
    BothPressed += 1;
else
    BothPressed = 0;
if (BothPressed >= 10)
{
    DoublePress = 1;
    DecPress = 0;
    IncPress = 0;
    BothPressed = 0;
}
// Defining decrement press
if ((dec == 1) && (inc == 0))
    DecPressed += 1;
else
    DecPressed = 0;
if (DecPressed >= 10)
{
    DoublePress = 0;
    DecPress = 1;
    IncPress = 0;
    DecPressed = 0;
}
```

```
// Defining increment press
if ((dec == 0) && (inc == 1))
    IncPressed += 1;
else
    IncPressed = 0;
if (IncPressed >= 10)
{
    DoublePress = 0;
    DecPress = 0;
    IncPress = 1;
    IncPressed = 0;
}
```

Snippet 4.5 | Defining button presses.

The DoublePress, DecPress, and IncPress values are Boolean state variables that raise flags when a button state has been pressed, or "raises the hand" of these variables so that this information can be processed later.

4.5. HX711 and K155ID1 Interface: Program Side

We have already discussed the ramifications of the HX711 and K155ID1 from a hardware perspective. We've also discussed the circuit design to interface with these components. Now, we enter the final step: writing code to push our four-byte binary integers to the Nixie tubes.

In Section 1, we defined three important pins on the HX711: the Latch Pin, the Data Pin, and the Clock Pin, the three pinouts that allow us to operate the shift register. The Data Pin, or Serial pin, accepts binary values; the STCP/Latch Pin defines whether the HX711 is listening for values; and the SHCP/Clock Pin shifts the value in the Serial/first register to the next available position on every rising edge transition. Our program initializes these three pins in the setup() function.

```
526    if (OpMode == 1)
527    {
528        DivNumber = 0;
529
530        dataH = GetBinary_ByteForm(T_h_ones, T_h_tens);
531        dataM = GetBinary_ByteForm(T_m_ones, T_m_tens);
532        dataS = GetBinary_ByteForm(T_s_ones, T_s_tens);
533
534        for (int j = 0; j < 1; j++)
535        {
536           digitalWrite(latchPin, 0); // Start listening
537           shiftOut(dataPin, clockPin, dataH);
538           shiftOut(dataPin, clockPin, dataM);
539           shiftOut(dataPin, clockPin, dataS);
540           digitalWrite(latchPin, 1); // Stop listening
541        }
542    }
```

Snippet 4.6 | Pushing binary values to the HX711.

The OpMode state variable determines whether the program is operating in State 1 or State 5: the hub/time state, or the message display state. When OpMode == 1, the program is in State 1.

The three variables dataS, dataM, and dataH each store a full byte of information, carrying the four-bit ones' and tens' values together.[8] After each ones-and-tens binary pair is concatenated, these values are sent through the `shiftOut()` command to be physically transferred to the registers in the HX711. Note that the Latch Pin must be set low before shifting values. When the Latch Pin undergoes a rising edge transition to high, the HX711 stops listening and the registers' 8-bit representation is shifted to physical representation on pins Qa-Qh.

The ShiftOut library [9] uses the Serial/Data Pin and the Clock Pin to shift values into the registers when the Latch Pin is listening. To generalize, the procedure goes as follows: the Clock Pin is set low to prepare for rising edge transition; the Serial Pin holds the next value to be passed through; the Clock Pin is set high so that the rising edge transition shifts the Data Pin values to the next register.

[8] The circuit schematic in Section 3 was accidentally wired in the wrong order, but this is not an issue. Due to the wiring configuration of the three HX711 chips, where the ESP32 Serial is connected to seconds first, the order of shifting is actually seconds, then minutes, then hours. In my *final* program code, lines 530 and 532 have the left-hand side of the equal sign switched in order to fix this.

4.6. Time-Responsive LEDs

I included yellow LEDs as separators for the hours, minutes, and seconds on the clock. Instead of having them always on, the LEDs undergo a “bounce” effect, transitioning from off, to on, then off again with a period of two seconds. This behavior is better explained by the following equation and graph.

$$PWM = \left|255 \cdot (\sin(\pi \cdot \frac{time}{200}))\right|$$

```
pulseglow = fabs( 255 * ( sin( (3.1415)*(((double)T_msec)/200) ) ) );
```

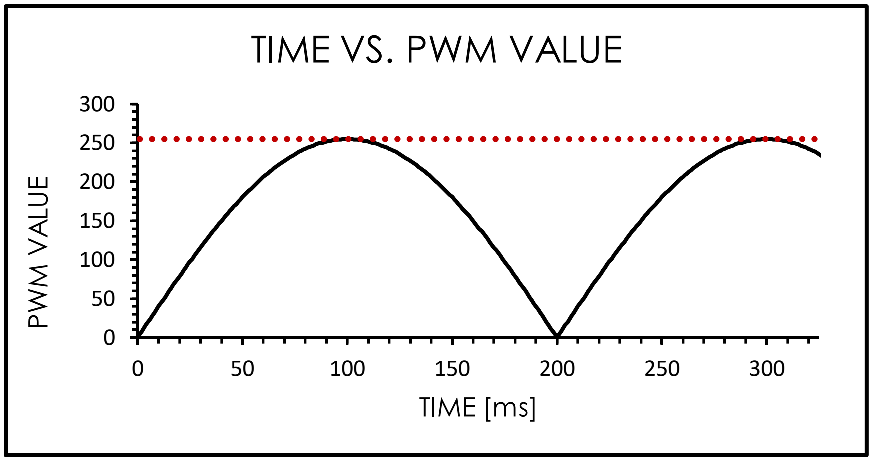


Figure 4.2 | Plot of PWM equation.

5. Additional Features

5.1. Music

I added music to my nixie tube clock as an alarm, with the option of also having it play at the top of the hour. Only one GPIO pin is required per buzzer. Each additional chord note requires another buzzer. For the example provided below in Figure 5.1, two buzzers would be required to play the melody. This song can also be played with a single buzzer, taking the high note of each chord.

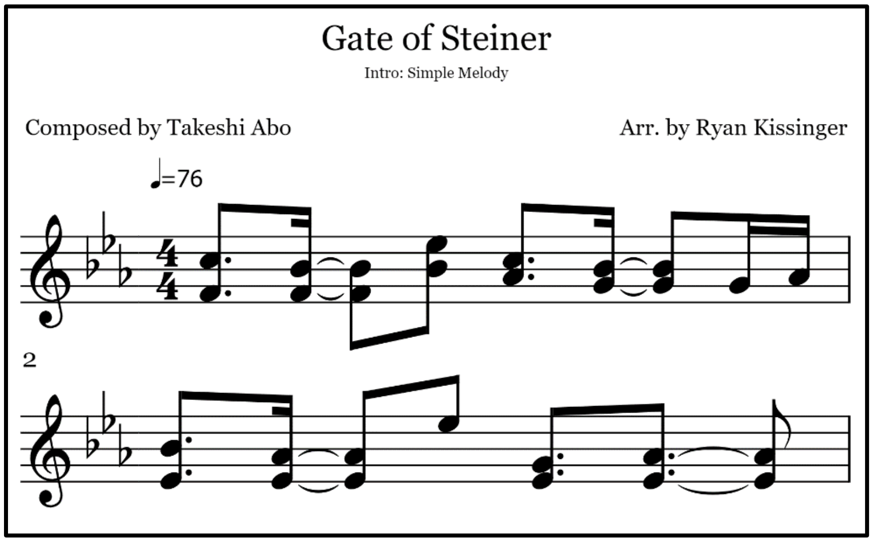


Figure 5.1 | Sample song for creating music with a buzzer.

I will cover the process of converting sheet music into a song for the buzzer. I assume the reader has minimum experience with music theory, so I will briefly explain the math behind calculating the rhythm of the song and turning these notes into code.

The notation ♪=76 expresses the beats-per-minute rhythm of the song. The stacked fours represent that there are four beats per measure, or per section ending with a vertical bar. So with this particular example, we derive the following equation to calculate beats per millisecond:

$$\left(\frac{76\,beats}{min}\cdot\frac{1\,min}{60\,sec}\cdot\frac{1\,sec}{1000\,ms}\right)^{-1} = \mathbf{789}\ \boldsymbol{\frac{ms}{beat}}$$

Now that the timing has been established, we must also know how to convert music notes into frequencies. Middle C has a frequency of

261.63. A lookup table of note vs. frequency (provided in Appendix D) presents us with the values of each note. To use my selected song as an example, this is the table I generated from the musical values. Note names follow the notation of note and octave: C5, for example, is octave 5 note C, one octave above middle C.

Note	Name	Frequency [hz]	Time from start [ms]
1	C5	523.25	0
	F4	349.23	
2	B^{b}4	466.16	591.75
	F4	349.23	
3	E^{b}5	659.25	1183.50
	B^{b}4	466.16	
4	C5	523.25	1578.00
	A^{b}4	415.30	
5	B^{b}4	466.16	2169.75
	G4	392.00	
6	G4	392.00	2761.50
	--	--	
7	A^{b}4	415.30	2958.75
	--	--	
8	B^{b}4	466.16	3156.00
	E^{b}4	311.13	
9	A^{b}4	415.30	3747.75
	E^{b}4	311.13	
10	E^{b}5	659.25	4339.50
	--	--	
11	G4	392.00	4734.00
	E^{b}4	311.13	
12	B^{b}4	466.16	5720.25
	E^{b}4	311.13	

Table 5.1 | Note frequency and timings for example song.

Since I used a single buzzer, my code uses only the topmost notes of each chord. My sample code can be found below. In the first code snippet, notes and timings are both defined in 12-value arrays.

```
double GOS_duration[13] =
{
	100*Gmult,
	591*Gmult,
	1183*Gmult,
	1578*Gmult,
	2169*Gmult,
	2761*Gmult,
	2958*Gmult,
	3156*Gmult,
	3747*Gmult,
	4339*Gmult,
	4734*Gmult,
	5420*Gmult,
	6000*Gmult
};
```

```
double GOS_highnote[13] =
{
	NOTE_C5,
	NOTE_AS4,
	NOTE_DS5,
	NOTE_C5,
	NOTE_AS4,
	NOTE_G4,
	NOTE_GS4,
	NOTE_AS4,
	NOTE_GS4,
	NOTE_DS5,
	NOTE_G4,
	NOTE_GS4,
	0
};
```

Snippet 5.1 | Defining notes and timings.

The code below implements the arrays we defined above. It plays the notes with respect to the process of our finite state machine, when the operation mode is set to play music.

```
if (OpMode == 2)
	{
		if (EstablishSongStart == false)
		{
			Serial.println("Song Started.");
			SongStart = currenttime;
			EstablishSongStart = true;
			NextNote = 0;
			SongTime = 0;
			ActiveNote = 0;
			digitalWrite(divPin, 1);
		}
		SongTime = currenttime - SongStart - 1000;
		if (SongTime > SongStart+GOS_duration[ActiveNote])
		{
			ledcWriteTone(2,GOS_highnote[ActiveNote]);
			ledcWrite(2,255);
			ActiveNote += 1;
		}
```

```
        if (ActiveNote > 14)
        {
               SongEnd = true;
               Serial.println("Song ended.");
        }
        if (SongEnd == true)
        {
               EstablishSongStart = false;
               SongEnd = false;
               OpMode = 1;
               ActiveNote = 0;
               NextNote = 0;
               ledcWrite(2,0);
               digitalWrite(divPin, 0);
        }
}
```

Snippet 5.2 | Example code for programming a song.

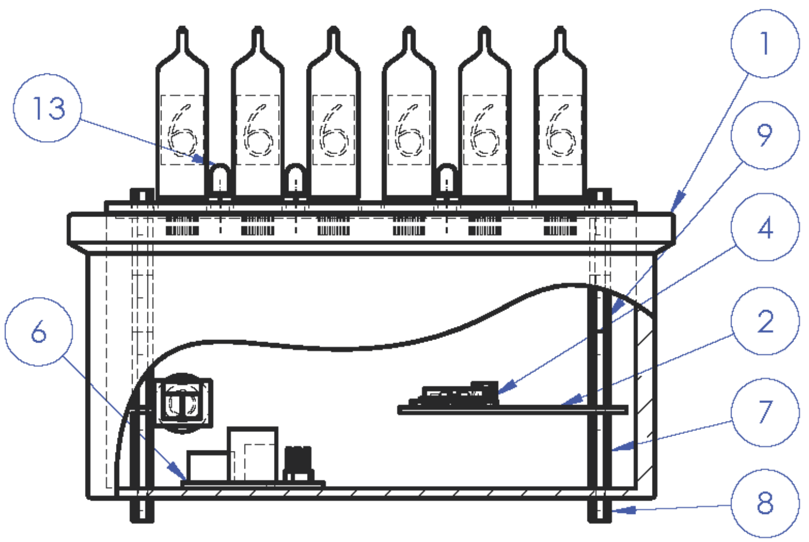
If the song has not yet been started, all the values are set to their respective initial quantities. Each pass through the main program code, the SongTime is calculated according to the time that has passed since the song began. Should the current time be greater than the value in the array defining note timings, the respective note is played, and the array is incremented to the next note. If the active note is greater than 14, the song is over, And if the song has ended, all the state variables are reset, and the sound is turned off.

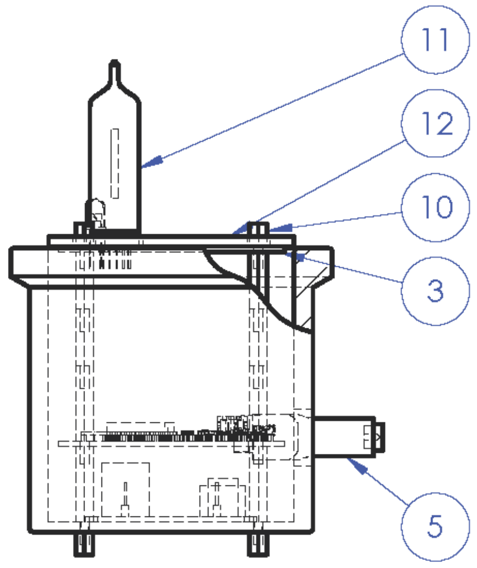
APPENDICES.

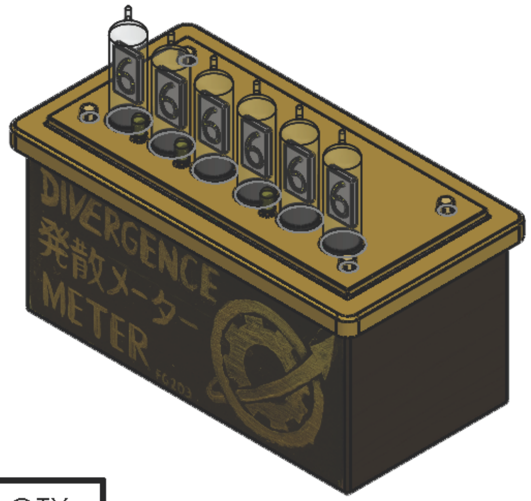
Appendix A: Interfacing with Sloeber

[WIP]

Appendix B: Engineering Drawings







1:2

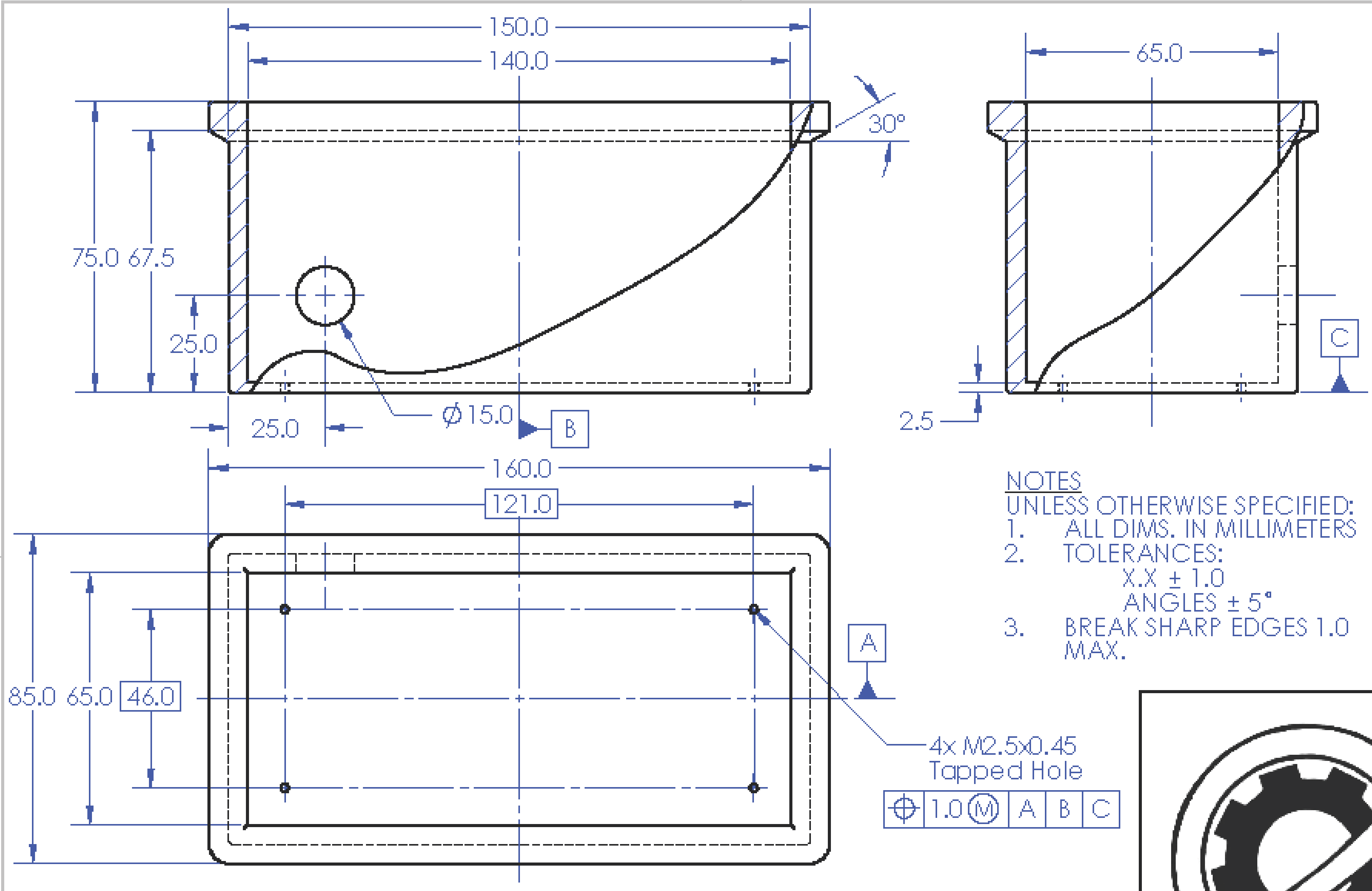
ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	DIVERGENCE METER BOX		1
2	BOTTOM PCB		1
3	TOP PCB		1
4	Adafruit HUZZAH32 ESP32 Feather		1
5	12V CONNECTOR		1
6	VOLTAGE STEP-UP		1
7	STANDOFFS	M2.5x20 + 6mm	8
8	STANDOFFS	M2.5x6mm	4
9	STANDOFFS	M2.5x15 + 6mm	8
10	STANDOFFS	M2.5x6 + 6mm	4
11	IN-16 NIXIE TUBE		6
12	TOP/COVER BOARD		1
13	YELLOW_LED		3

NOTES

CONFIRM TEST DATA WITH PHONEWAVE - READINGS POSITIVE AND MAKISE TIME DILATION EFFECT. PUSH NEW DIMENSIONS TO D.A.S.H.
電話でテストデータを確認する-
朗読、正解および時間短縮
効果。 D.A.S.H.に新しい次元をプッシュ

DWG NO.	1 OF 8
VERSION	1.0
PART	DIVERGENCE METER
AUTHOR(S)	岡部 倫太郎 (LM 001) RYAN KISSINGER

FUTURE GADGET LAB
WESTERN DIVISION - SITE 117



NOTES
UNLESS OTHERWISE SPECIFIED:
1. ALL DIMS. IN MILLIMETERS
2. TOLERANCES:
X.X ± 1.0
ANGLES ± 5°
3. BREAK SHARP EDGES 1.0 MAX.

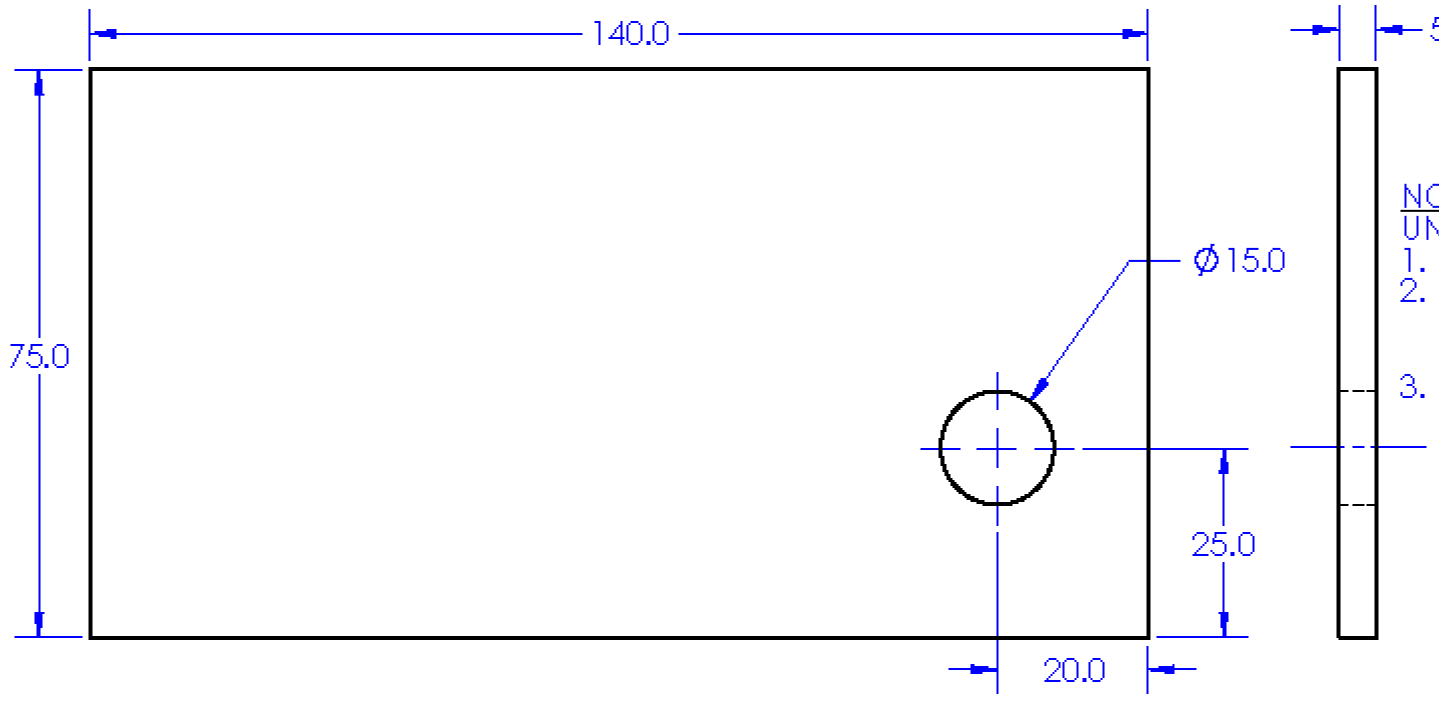
2:3

NOTES

CONFIRM TEST DATA WITH PHONEWAVE - READINGS POSITIVE AND MAKISE TIME DILATION EFFECT. PUSH NEW DIMENSIONS TO D.A.S.H.
電話でテストデータを確認する-
朗読、正解および時間短縮
効果。 D.A.S.H.に新しい次元をプッシュ

DWG NO.	2 OF 8
VERSION	1
PART	BOX ENCLOSURE
AUTHOR(S)	岡部 倫太郎 (LM 001) RYAN KISSINGER

FUTURE GADGET LAB
WESTERN DIVISION - SITE 117



NOTES
UNLESS OTHERWISE SPECIFIED:
1. ALL DIMS. IN MILLIMETERS
2. TOLERANCES:
X.X ± 1.0
ANGLES ± 5°
3. BREAK SHARP EDGES 1.0 MAX.

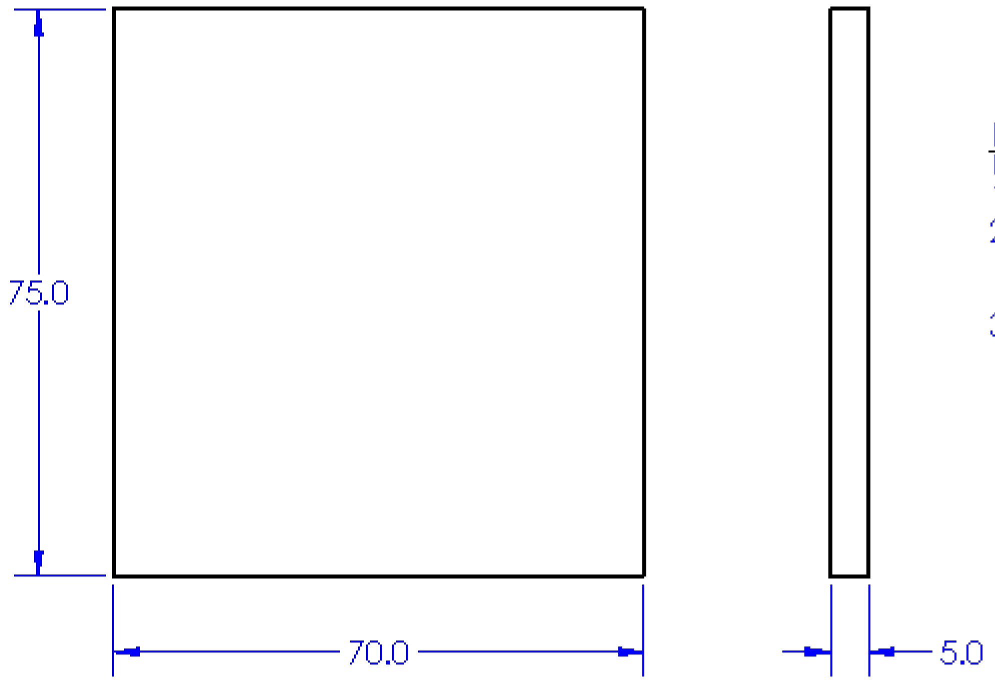
1:1

NOTES	DWG NO.	3 OF 8
CONFIRM TEST DATA WITH PHONEWAVE - READINGS POSITIVE AND MAKISE TIME DILATION EFFECT. PUSH NEW DIMENSIONS TO D.A.S.H. 電話でテストデータを確認する- 朗読、正解および時間短縮 効果。 D.A.S.H.に新しい次元をプッシュ	VERSION	1
	PART	BOARD, BACK
	AUTHOR(S)	岡部 倫太郎 (LM 001) RYAN KISSINGER

FUTURE GADGET LAB
WESTERN DIVISION - SITE 117

2 1

B B



NOTES
UNLESS OTHERWISE SPECIFIED:
1. ALL DIMS. IN MILLIMETERS
2. TOLERANCES:
   X.X ± 1.0
   ANGLES ± 5°
3. BREAK SHARP EDGES 1.0 MAX.

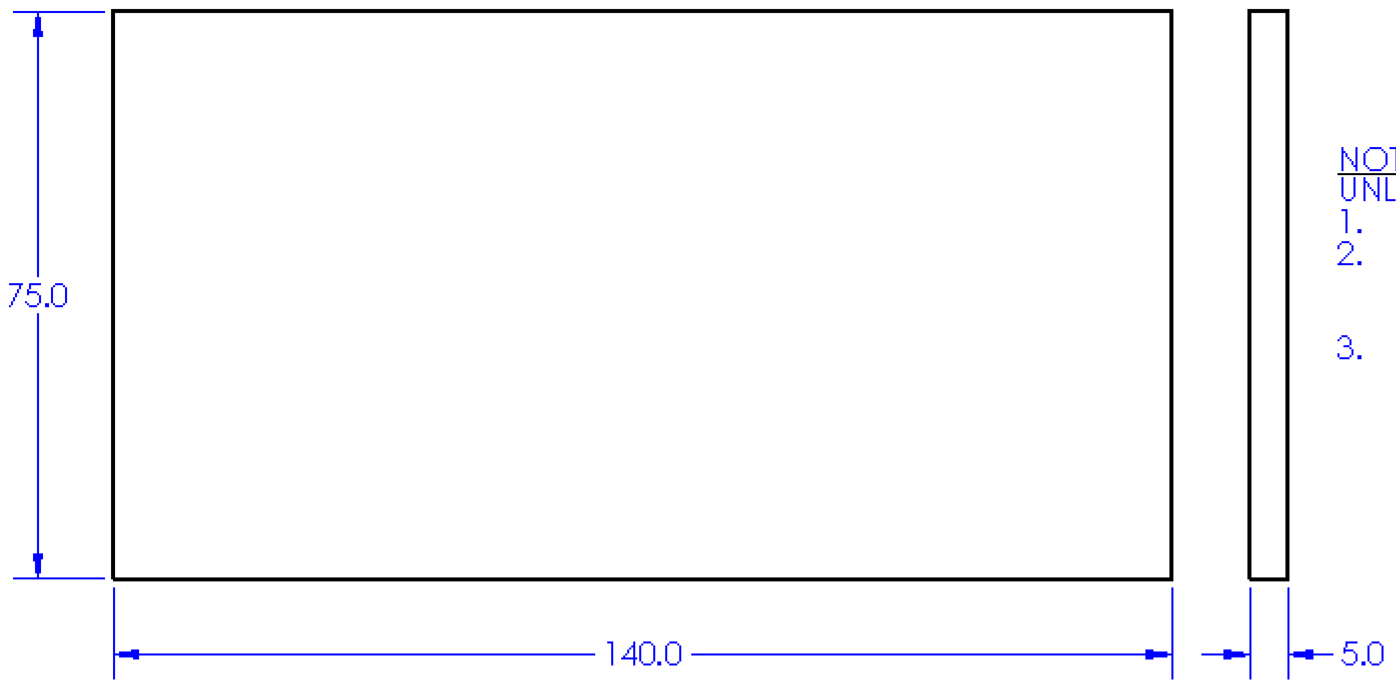
1:1

A A

NOTES CONFIRM TEST DATA WITH PHONEWAVE - READINGS POSITIVE AND MAKISE TIME DILATION EFFECT. PUSH NEW DIMENSIONS TO D.A.S.H. 電話でテストデータを確認する- 朗読、正解および時間短縮 効果。 D.A.S.H.に新しい次元をプッシュ	DWG NO.	4 OF 8
	VERSION	1
	PART	BOARD, LT + RT
	AUTHOR(S)	岡部 倫太郎 (LM 001) RYAN KISSINGER

FUTURE GADGET LAB
WESTERN DIVISION - SITE 117

2 1

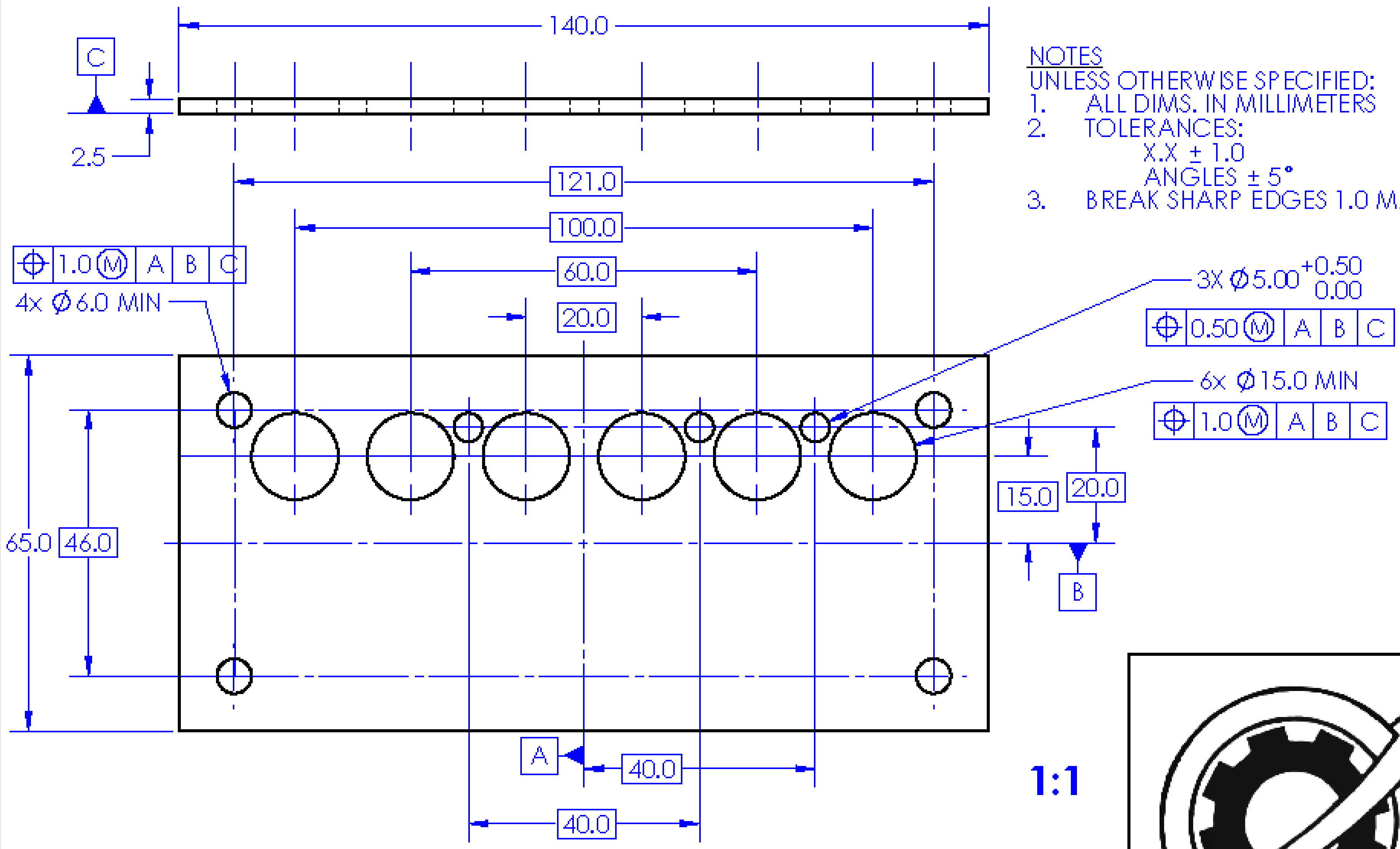


NOTES
UNLESS OTHERWISE SPECIFIED:
1. ALL DIMS. IN MILLIMETERS
2. TOLERANCES:
X.X ± 1.0
ANGLES ± 5°
3. BREAK SHARP EDGES 1.0 MAX.

1:1

NOTES		
CONFIRM TEST DATA WITH PHONEWAVE - READINGS POSITIVE AND MAKISE TIME DILATION EFFECT. PUSH NEW DIMENSIONS TO D.A.S.H. 電話でテストデータを確認する- 朗読、正解および時間短縮 効果。 D.A.S.H.に新しい次元をプッシュ	DWG NO.	5 OF 8
	VERSION	1
	PART	BOARD, FRONT
	AUTHOR(S)	岡部 倫太郎 (LM 001) RYAN KISSINGER

FUTURE GADGET LAB
WESTERN DIVISION - SITE 117



NOTES
UNLESS OTHERWISE SPECIFIED:
1. ALL DIMS. IN MILLIMETERS
2. TOLERANCES:
X.X ± 1.0
ANGLES ± 5°
3. BREAK SHARP EDGES 1.0 MAX.

1:1

NOTES

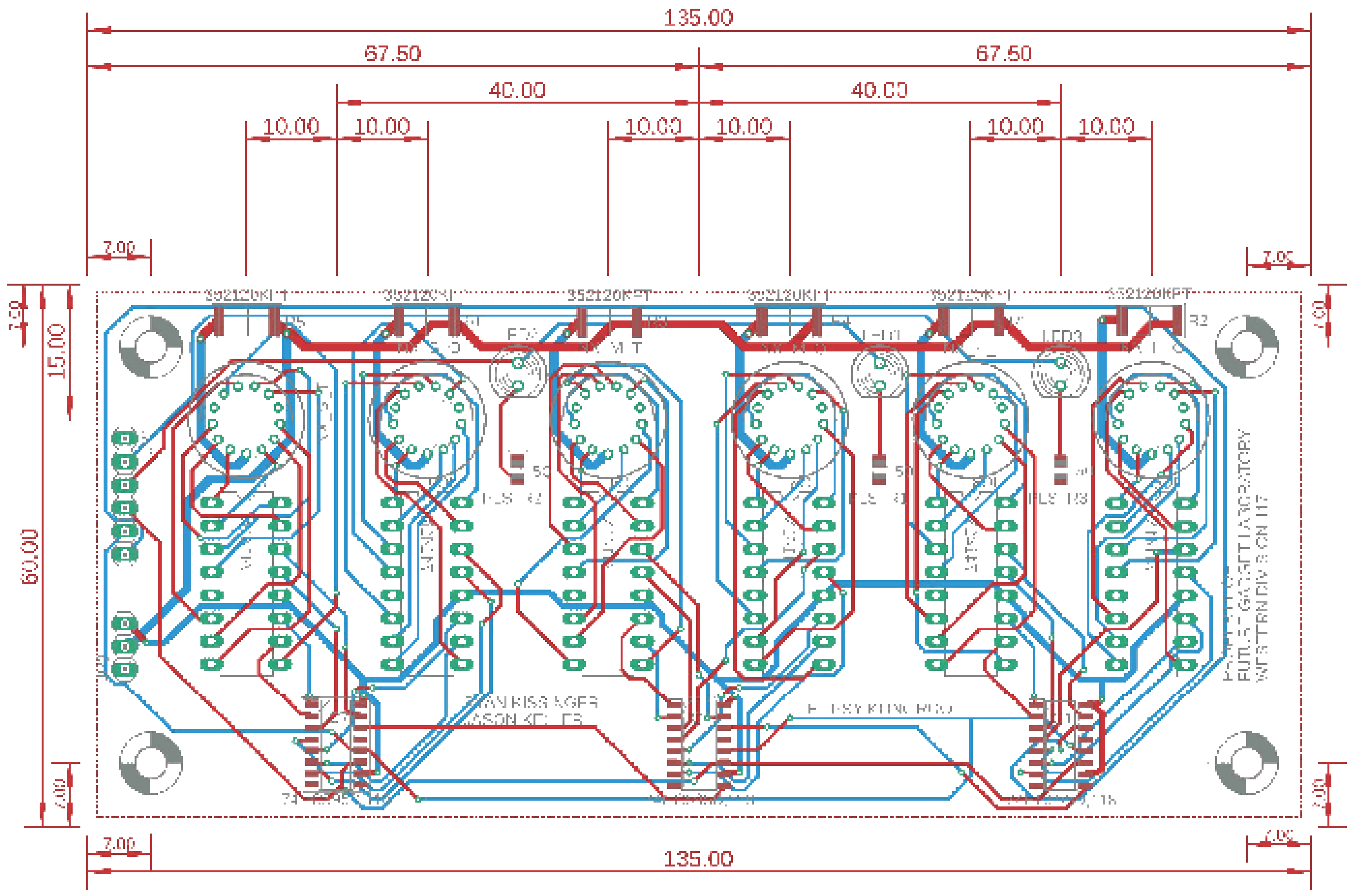
CONFIRM TEST DATA WITH PHONEWAVE - READINGS POSITIVE AND MAKISE TIME DILATION EFFECT. PUSH NEW DIMENSIONS TO D.A.S.H.
電話でテストデータを確認する-
朗読、正解および時間短縮
効果。 D.A.S.H.に新しい次元をプッシュ

DWG NO.	6 OF 8
VERSION	1
PART	TOP/COVER BOARD
AUTHOR(S)	岡部 倫太郎 (LM 001) RYAN KISSINGER

FUTURE GADGET LAB
WESTERN DIVISION - SITE 117

2 1

B B



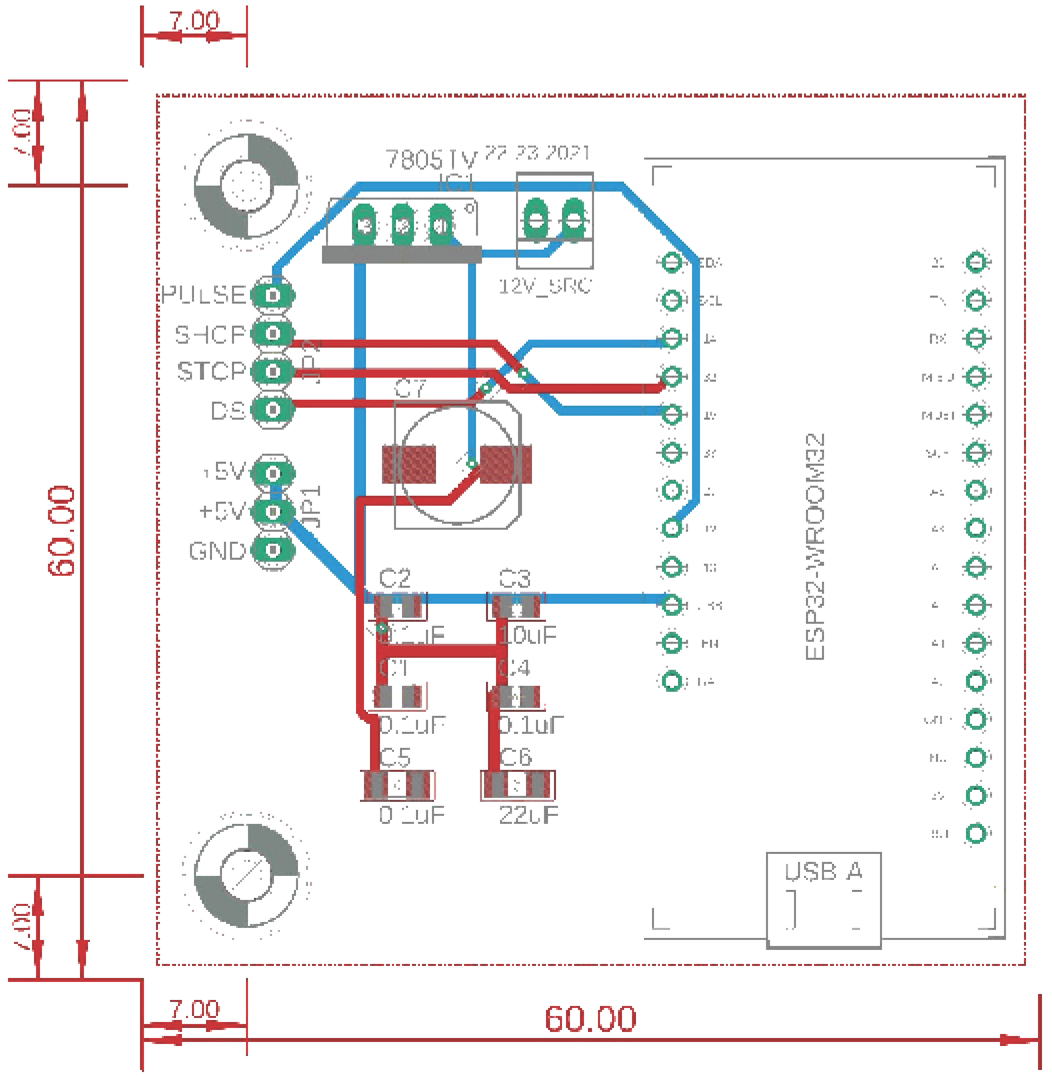
A A

NOTES	DWG NO.	7 OF 8
CONFIRM TEST DATA WITH PHONEWAVE - READINGS POSITIVE AND MAKISE TIME DILATION EFFECT. PUSH NEW DIMENSIONS TO D.A.S.H. 電話でテストデータを確認する- 朗読、正解および時間短縮 効果。 D.A.S.H.に新しい次元をプッシュ	VERSION	1
	PART	UPPER PCB
	AUTHOR(S)	岡部 倫太郎 (LM 001) RYAN KISSINGER

FUTURE GADGET LAB

WESTERN DIVISION - SITE 117

2 1

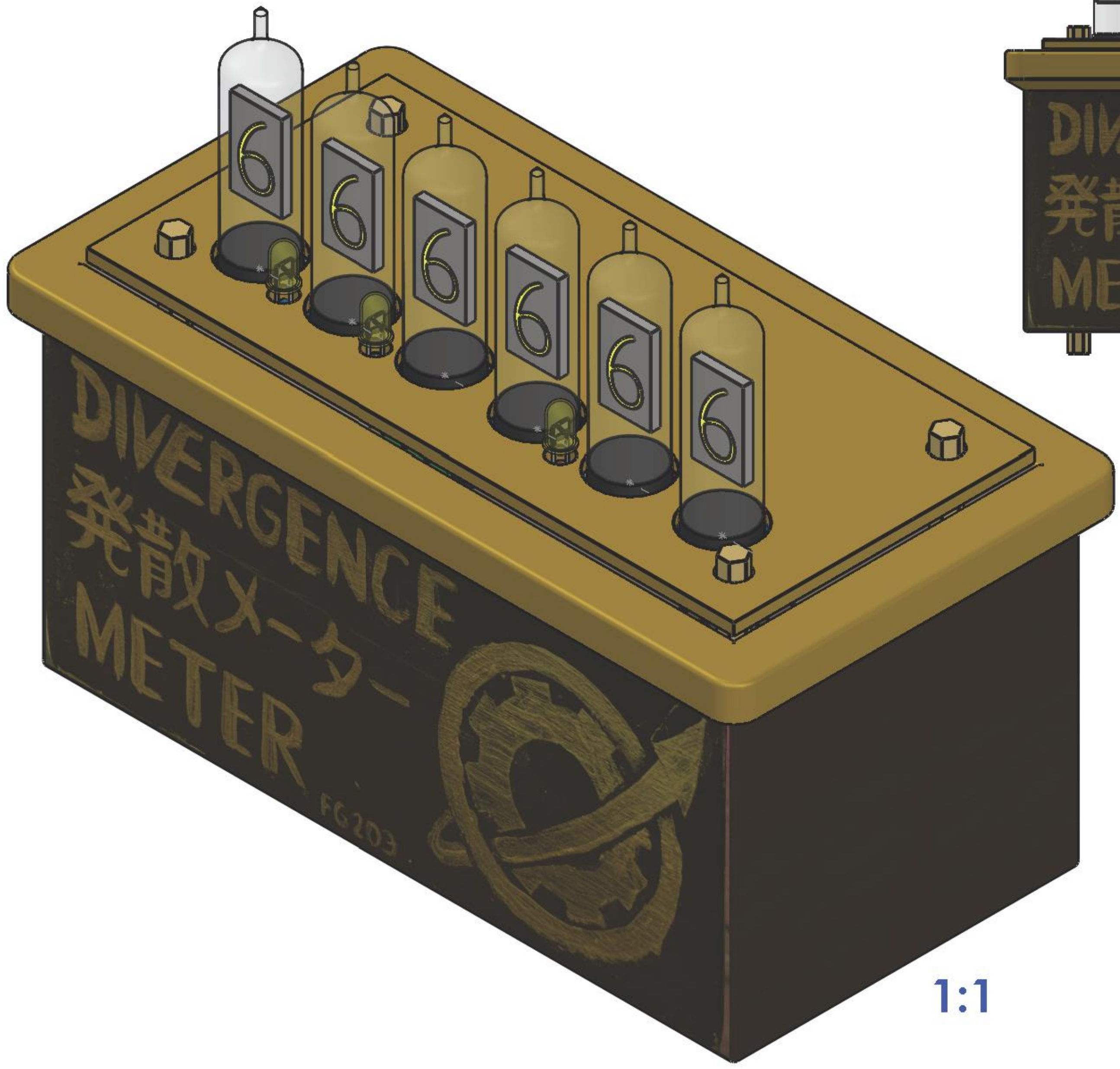


NOTES	DWG NO.	8 OF 8
CONFIRM TEST DATA WITH PHONEWAVE - READINGS POSITIVE AND MAKISE TIME DILATION EFFECT. PUSH NEW DIMENSIONS TO D.A.S.H. 電話でテストデータを確認する- 朗読、正解および時間短縮 効果。 D.A.S.H.に新しい次元をプッシュ	VERSION	1
	PART	LOWER PCB
	AUTHOR(S)	岡部 倫太郎 (LM 001) RYAN KISSINGER

FUTURE GADGET LAB
WESTERN DIVISION - SITE 117

2 1

B B



4:7

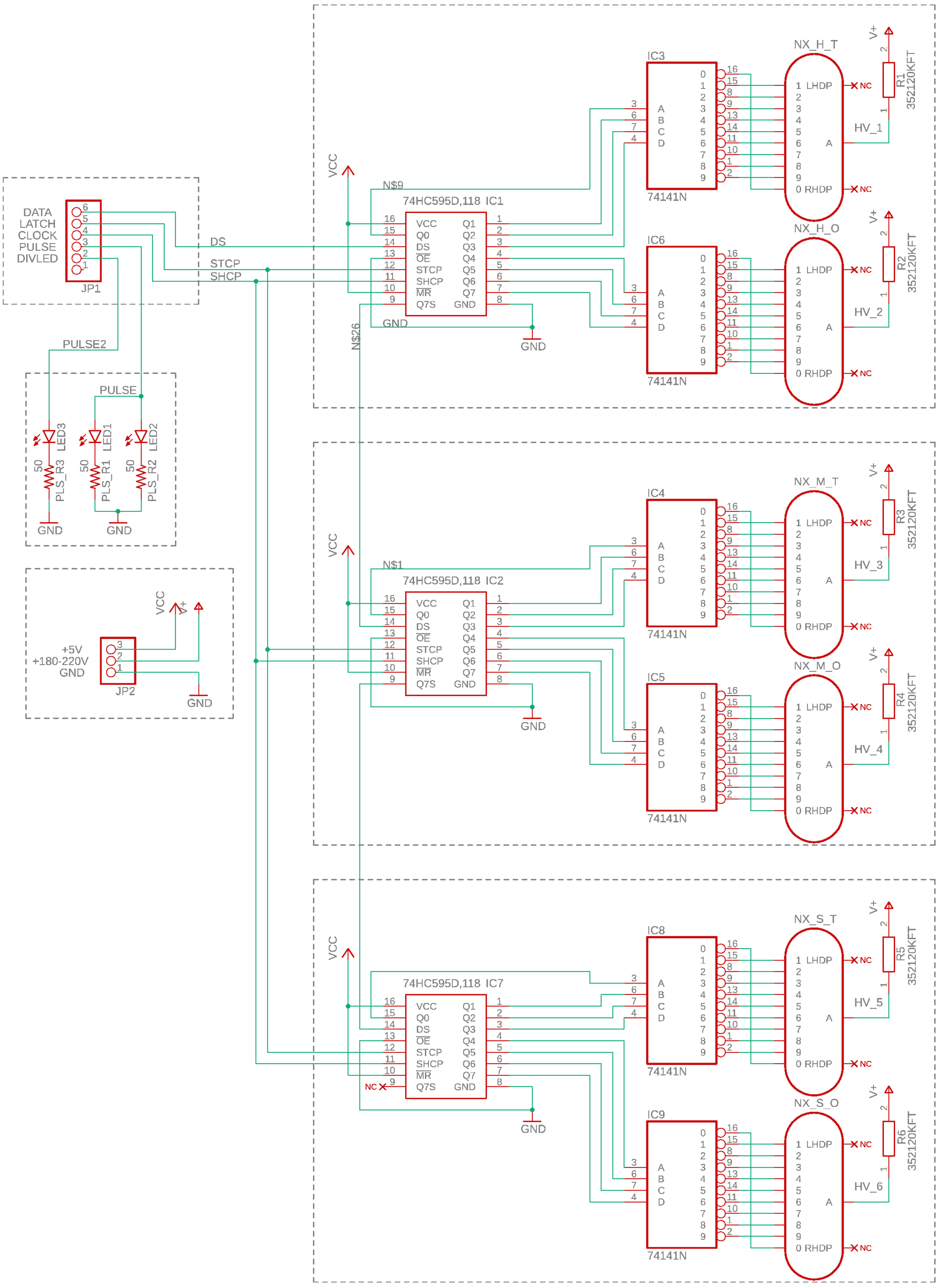
1:1

A A

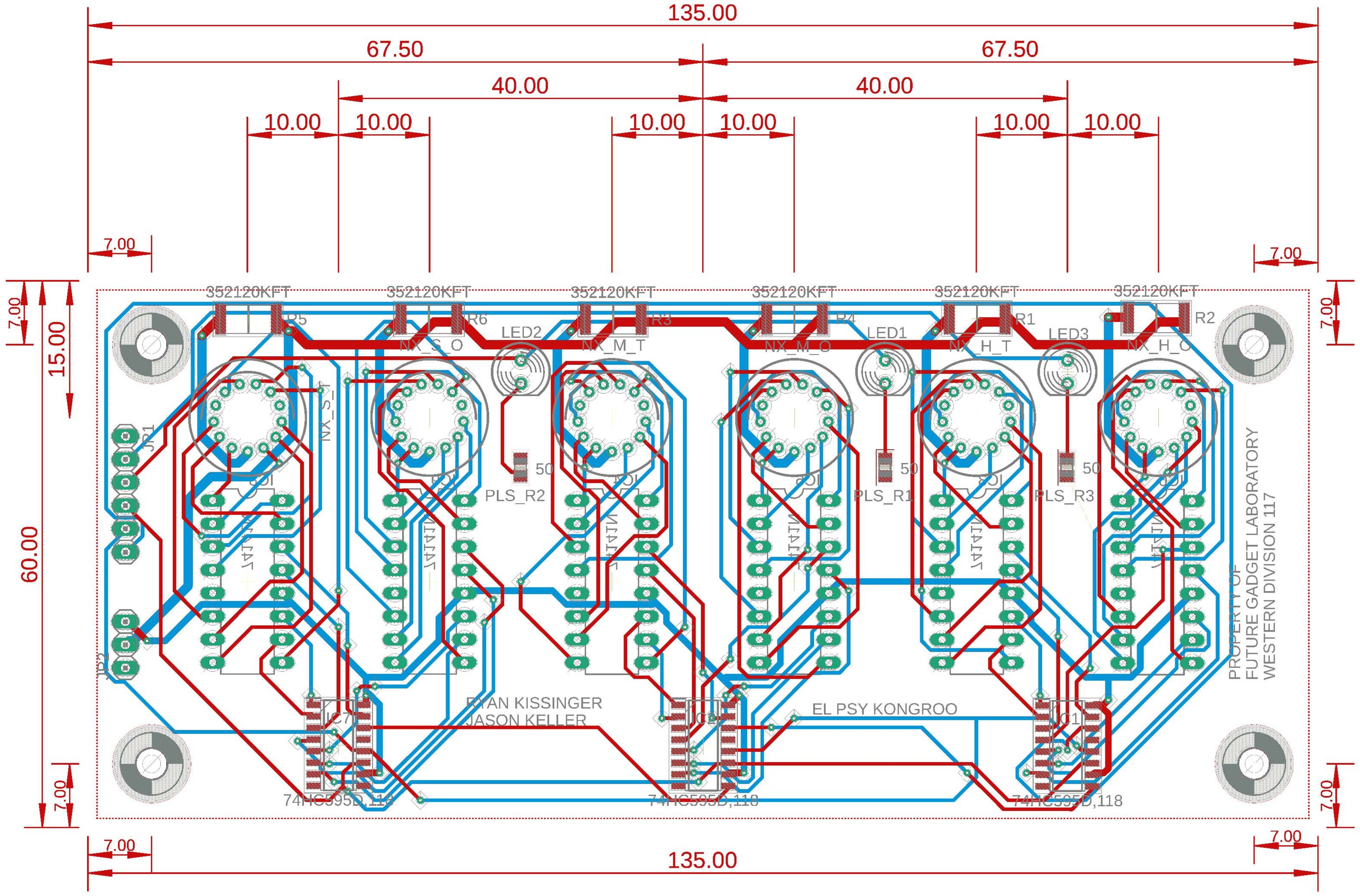
FUTURE GADGET LAB
WESTERN DIVISION - SITE 117

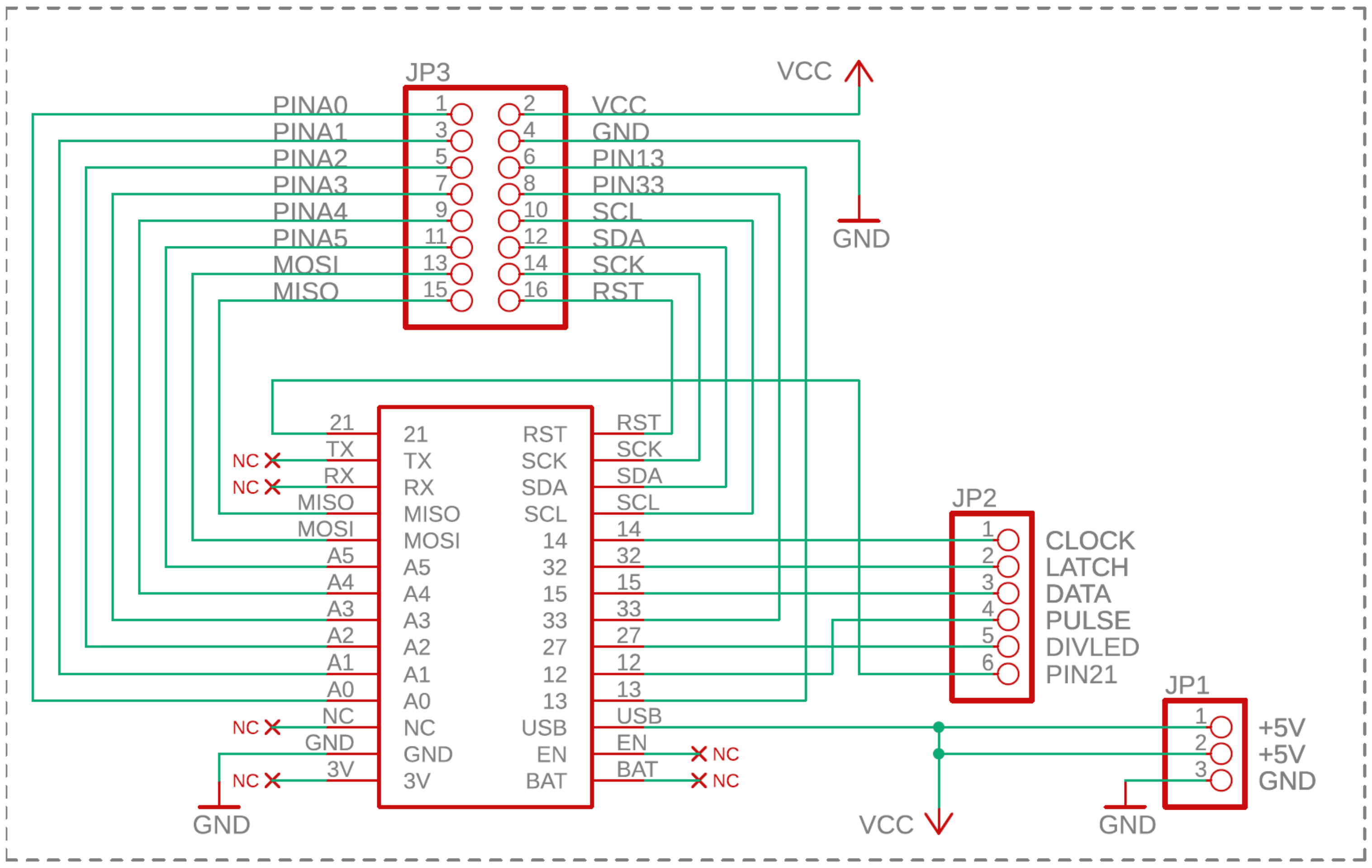
2 1

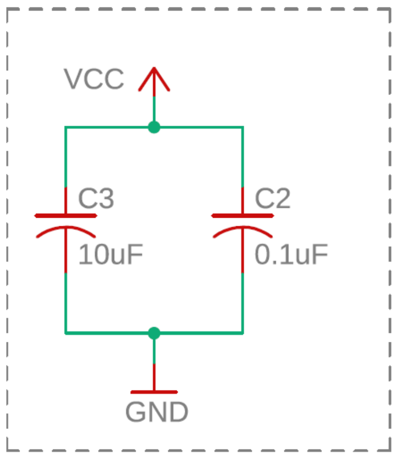
Appendix C: Electrical Schematics

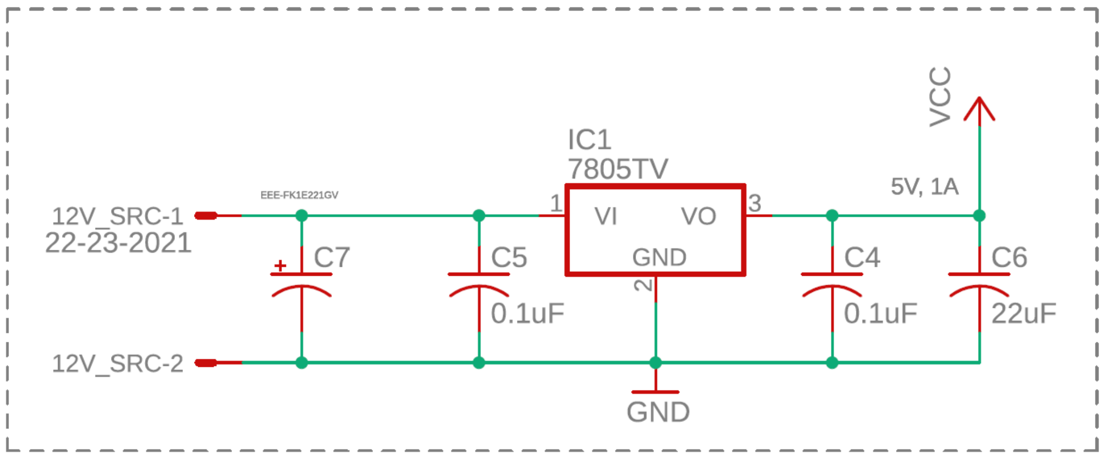


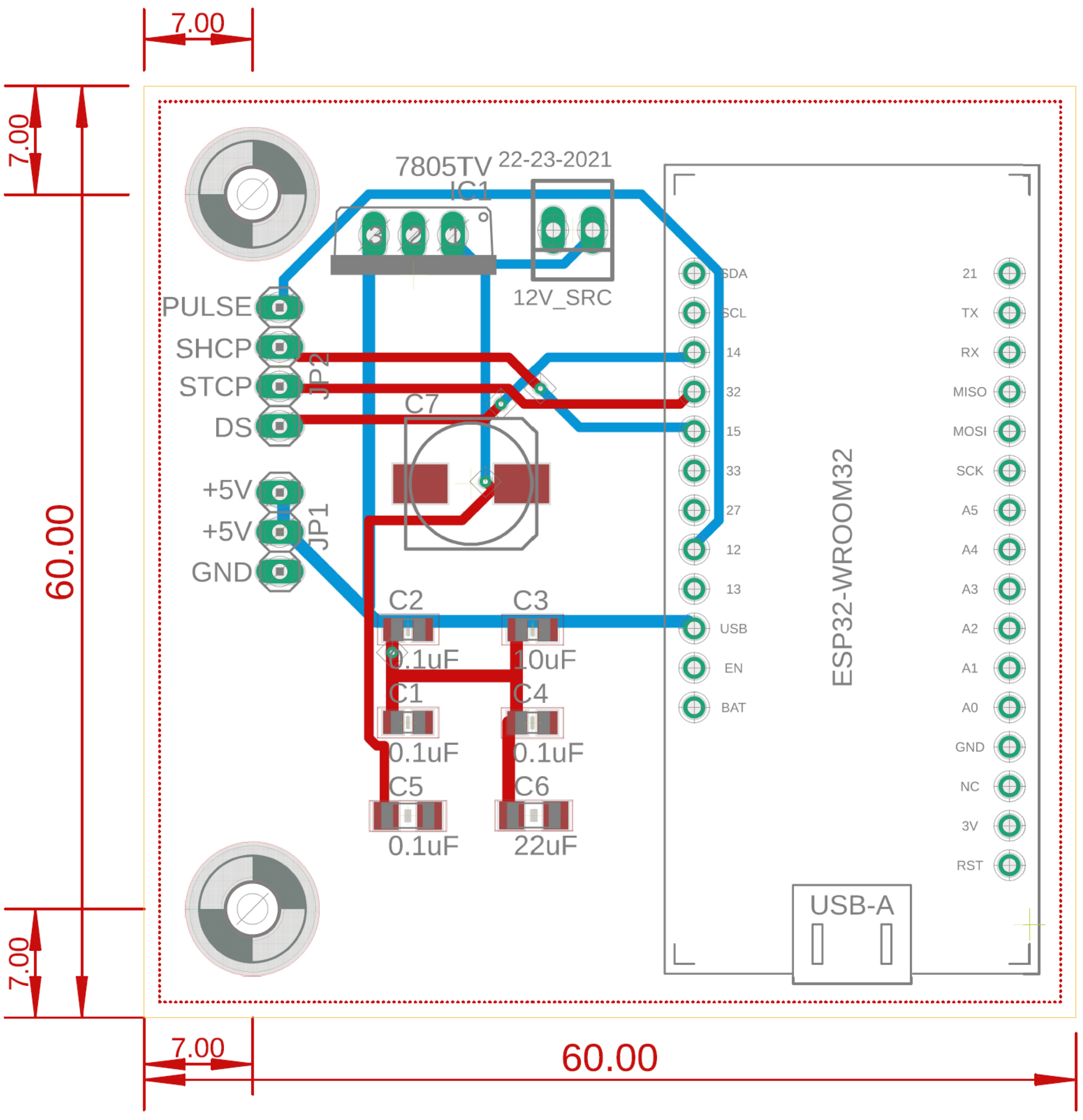
DATA
LATCH
CLOCK
PULSE
DIVLED
JP1
DS
STCP
SHCP
PULSE2
PULSE
LED1
LED2
LED3
PLS_R1
PLS_R2
PLS_R3
GND
VCC
V+
+5V
+180-220V
GND
JP2
74HC595D,118 IC1
74HC595D,118 IC2
74HC595D,118 IC7
N$9
N$1
N$26
74141N
IC3
IC4
IC5
IC6
IC8
IC9
NX_H_T
NX_H_O
NX_M_T
NX_M_O
NX_S_T
NX_S_O
HV_1
HV_2
HV_3
HV_4
HV_5
HV_6
R1
R2
R3
R4
R5
R6
352120KFT
LHDP
RHDP
NC


135.00
67.50
67.50
40.00
40.00
10.00
10.00
10.00
10.00
10.00
10.00
7.00
7.00
7.00
7.00
7.00
7.00
7.00
7.00
15.00
60.00
135.00
352120KFT
R5
R6
R3
R4
R1
R2
LED2
LED1
LED3
NX_S_O
NX_M_T
NX_M_O
NX_H_T
NX_H_O
NX_S_T
JP1
JP2
PLS_R2
PLS_R1
PLS_R3
50
IC7
IC1
74HC595D,118
RYAN KISSINGER
JASON KELLER
EL PSY KONGROO
PROPERTY OF
FUTURE GADGET LABORATORY
WESTERN DIVISION 117


JP3
VCC
PINA0
PINA1
PINA2
PINA3
PINA4
PINA5
MOSI
MISO
VCC
GND
PIN13
PIN33
SCL
SDA
SCK
RST
GND
21
TX
RX
MISO
MOSI
A5
A4
A3
A2
A1
A0
NC
GND
3V
RST
SCK
SDA
SCL
14
32
15
33
27
12
13
USB
EN
BAT
NC
JP2
CLOCK
LATCH
DATA
PULSE
DIVLED
PIN21
JP1
+5V
+5V
GND
VCC
GND
GND


VCC
C3
10uF
C2
0.1uF
GND


IC1
7805TV
VCC
5V, 1A
12V_SRC-1
22-23-2021
EEE-FK1E221GV
C7
C5
0.1uF
VI
VO
GND
C4
0.1uF
C6
22uF
12V_SRC-2
GND


7.00
7.00
60.00
7.00
7.00
60.00
7805TV
22-23-2021
IC1
12V_SRC
PULSE
SHCP
STCP
DS
JP2
C7
+5V
+5V
GND
JP1
C2
0.1uF
C3
10uF
C1
0.1uF
C4
0.1uF
C5
0.1uF
C6
22uF
SDA
SCL
14
32
15
33
27
12
13
USB
EN
BAT
21
TX
RX
MISO
MOSI
SCK
A5
A4
A3
A2
A1
A0
GND
NC
3V
RST
ESP32-WROOM32
USB-A

Appendix D: Full Program Code

Click this link for the attached .ino file:

ElPsyKongroo.ino

```
#include "Arduino.h"
#include <stdio.h>
#include <math.h>

//====================================================================================
      //
//      NAME          :      Divergence Shifter Clock
      //
//                    :      FUTURE GADGET LABS, Western Office
      //
//                    :
      //
//      AUTHOR        :      Ryan Kissinger, Jason Keller
      //
//                    :      r.kissinger68@gmail.com
      //
//                    :      https://autononymous.github.io/index.html/
      //
//                    :
      //
//      DATE          :      16 January 2020
      //
//                    :
      //
//      VERSION       :      2.31 (First)
      //
//                    :
      //
//      DEVICE        :      ESP32-WROOM32 MODULE (Adafruit)
      //
//                    :
      //
//====================================================================================
      //
//      NOTES         :             [ MODE 1A: CLOCK ]
      //
//                    :      Clock section runs on the millis() function, which
      //
//                    :      keeps an active tabulation of the time since the
      //
//                    :      program has started running. Seconds are converted
      //
//                    :      into minutes, then hours, and each digit is extracted
      //
//                    :      and converted into binary, for use in the 74HC595
      //
//                    :      Shift Register module.
      //
//                    :
      //
//                    :             [MODE 1B: DIVERGENCE HOUR CHIME]
      //
//                    :      Are you on the right worldline? At the top of each
      //
//                    :      hour, Gate of Steiner plays and the divergence meter
      //
```

```
//                  :      shows the current worldline.
      //
//                  :
      //
//                  :             [ MODE 2: CONSTANT DIVERGENCE]
      //
//                  :      When a connected switch is TRUE, the divergence meter
      //
//                  :      is constantly shown. (To be added)
      //
//                  :
      //
//                  :             [ LICENSE ]
      //
//                  :      This file is Copyright 2020 by Ryan M. Kissinger and
      //
//                  :      released under the Lesser GNU Public License, version
      //
//                  :      2. It intended for educational use only, but its use
      //
//                  :      is not limited thereto.
      //
//                  :             ----------------------------------------------
      //
//                  :      THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
      //
//                  :      CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRAN-
      //
//                  :      TIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WAR-
      //
//                  :      RANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
      //
//                  :      PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
      //
//                  :      OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIR-
      //
//                  :      ECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
      //
//                  :      DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
      //
//                  :      SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
      //
//                  :      PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
      //
//                  :      ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
      //
//                  :      LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
      //
//                  :      ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
      //
//                  :      EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
      //
//                  :
      //
//==================================================================================
      //
//    PINOUTS       :      32     -      Latch Pin, or STCP on the 74HC595.
      //
//                  :      14     -      Clock Pin, or SHCP on the 74HC595.
      //
//                  :      15     -      Data Pin, or DS on the 74HC595.
      //
```

```
//                  :      12     -      Pulse Pin, driving the two top-mounted LEDs
       //
//                  :                    that act as separators for the hour, minute,
       //
//                  :                    and second places.
       //
//                  :      XX     -      Divergence Pin, driving the LED separating the
       //
//                  :                    ones place of the active worldine.
       //
//                  :      27     -      Buzzer Pin, driving the speaker.
       //
//                  :
       //
//==================================================================================
       //
//     CREDITS      :             [ shiftOut MODULE 1.0 ]
       //
//                  :      Carilyn Maw & Tom Igoe
       //
//                  :      25 October 2006
       //
//==================================================================================
       //
//     RELEVANT DOCUMENTATION is listed at the bottom of this code.
       //
//==================================================================================
       //



//==================================================================================
===|
//===========================[ VARIABLE DEFINITIONS
]==================================|
//==================================================================================
===|

//-----[Defining music note frequencies]-----//

#define NOTE_C4  262       // "middle C"
#define NOTE_CS4 277
#define NOTE_D4  294
#define NOTE_DS4 311
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_FS4 370
#define NOTE_G4  392
#define NOTE_GS4 415
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_B4  494
#define NOTE_C5  523
#define NOTE_CS5 554
#define NOTE_D5  587
#define NOTE_DS5 622
#define NOTE_E5  659
#define NOTE_F5  698
#define NOTE_FS5 740
#define NOTE_G5  784
#define NOTE_GS5 831
#define NOTE_A5  880
#define NOTE_AS5 932
```

```
#define NOTE_B5  988


//-----[Defining music note frequencies]-----//
static int latchPin        = 32;
static int clockPin        = 14;
static int dataPin         = 15;
static int pulsePin        = 12;
static int buzzerPin       = 21;
static int incPin          = 26; // or Pin A0
static int decPin          = 25; // or Pin A1
static int statusPin       = 99; // Status of time change mode (27)
static int divPin          = 27;


static int timebuf         = 1;

//-----[Time variables]----------------------//
//     Below are the binary values for time.
//
//                                                                    TENS   ONES
byte dataH;                           // Hour                   XXXX   XXXX
byte dataM;                           // Minute           XXXX   XXXX
byte dataS;                           // Second           XXXX   XXXX


int currenttime;                      //     Time extracted from the millis()
function.
double mil[2];                        //     Records previous and last time state.
double d_mil;                         //     Calculates time change from (mil[1] -
mil[0]).

double T_msec;                        //     Time in milliseconds. Used for pulsing
of LEDs and music score timing.
int pulseglow;                        //     Holds the active value for PWM cycle of
the blinking time separator LEDs.

double T_sec_act;                     //     Active time in seconds.
double T_secs;                        //     Time in seconds:  {0 <= T_secs <= 59}.
int T_s_ones;                         //     Seconds ones' place.
int T_s_tens;                         //     Seconds tens' place.

double T_mins;                        //     Time in minutes:  {0 <= T_mins <= 59}.
int T_m_ones;                         //     Minutes ones' place.
int T_m_tens;                         //     Minutes tens' place.

double T_hrs;                         //     Time in hours:  {0 <= T_hrs <= 23}.
int T_h_ones;                         //     Hours ones' place.
int T_h_tens;                         //     Hours tens' place.

double T_days;                        //     Counter for number of days passed/
double Gmult = 1;                     //     Timer that slows or accelerates BPM of
song.


//-----[GATE OF STEINER Notes]-----------------//
//     4/4 time , 76 BPM or 789ms/beat
double GOS_duration[13] =
{100*Gmult,591*Gmult,1183*Gmult,1578*Gmult,2169*Gmult,2761*Gmult,2958*Gmult,3156*Gmult
,3747*Gmult,4339*Gmult,4734*Gmult,5420*Gmult,6000*Gmult};
//                             C5      Bb4      Eb5      C5      Bb4      G4      Ab4
Bb4      Ab4      Eb5      G4      Ab4
```

```
double GOS_highnote[13] =
{NOTE_C5,NOTE_AS4,NOTE_DS5,NOTE_C5,NOTE_AS4,NOTE_G4,NOTE_GS4,NOTE_AS4,NOTE_GS4,NOTE_DS
5,NOTE_G4,NOTE_GS4,0};

//-----[Song variables]------------------------//
bool EstablishSongStart;   //     Boolean returning if song has started.
bool SongEnd;                     //     Returns of song has ended or not
double SongStart;                 //     Start time calculated from when first run
double SongTime;                  //     Song time relative to the start of the song
double NextNote;                  //     Unused
double NoteTime;                  //     Value pulled from GOS_duration[]
int ActiveNote;                           //     Which note is currently active

//-----[Divergence Mode]-----------------------//
// 0-9 regular numbers, 10-16 nothing
int DivMode_SO[20] = { 16 , 16 , 16 , 16 ,  8 ,  1 ,  6 ,  2 ,  5 ,  2 ,  2 ,  1 ,  0
,  4 ,  1 ,  7 ,  0 ,  2 ,  3 ,  1 };
int DivMode_ST[20] = { 16 , 16 , 16 , 16 ,  9 ,  7 ,  6 ,  4 ,  1 ,  9 ,  1 ,  0 ,  6
,  2 ,  2 ,  1 , 16 ,  8 ,  1 ,  0 };
int DivMode_MO[20] = { 16 , 16 ,  2 ,  5 ,  3 ,  0 ,  6 ,  8 ,  6 ,  4 ,  6 ,  1 ,  5
,  0 ,  5 ,  4 ,  8 ,  5 ,  7 ,  4 };
int DivMode_MT[20] = { 16 , 16 , 16 ,  2 ,  2 ,  2 ,  6 ,  3 ,  0 ,  3 ,  7 ,  0 ,  5
,  6 ,  4 ,  2 , 16 ,  4 ,  3 ,  8 };
int DivMode_HO[20] = { 16 , 16 ,  4 ,  7 , 16 ,  5 ,  6 ,  0 ,  8 ,  3 ,  6 ,  1 ,  3
,  9 ,  9 ,  6 , 16 ,  3 ,  6 ,  5 };
int DivMode_HT[20] = { 16 ,  1 ,  8 ,  3 ,  4 ,  1 ,  6 ,  1 ,  3 ,  0 ,  7 ,  0 ,  6
,  6 ,  0 ,  3 ,  9 ,  1 ,  8 ,  9 };

int DivNumber = 0;
//-----[Program variables]---------------------//
int OpMode;                               //     Determines whether showing time or
playing song

int BothPressed;
int DecPressed;
int IncPressed;
int TimeMode;
bool ChangeTime;
bool DoublePress;
bool DecPress;
bool IncPress;
double PressButtonBuf;
int T_0 = 0;
bool SecondPassed = false;
double T_ms0 = 0;
bool MSPassed = false;
double currentms = 0;

//=====================================================================================
===|
//=================================[ MAIN PROGRAM
]=====================================|
//=====================================================================================
===|

void setup()
{
	pinMode(latchPin, OUTPUT);
	pinMode(clockPin, OUTPUT);
	pinMode(dataPin,  OUTPUT);
	pinMode(divPin, OUTPUT);
	pinMode(33,OUTPUT);
	ledcSetup(1,500,8);
	ledcSetup(2,200,8);
	ledcAttachPin(pulsePin, 1);
	ledcAttachPin(buzzerPin,2);
	pinMode(incPin, INPUT);
	pinMode(decPin, INPUT);
	pinMode(statusPin, OUTPUT);

	Serial.begin(9600);

//                            Hour      Minute    Second
	T_sec_act = (11*3600) + (20*60)  +  (02);      // This is the starting time for
the clock at boot.
	mil[0] = 0;
	T_days = 0;
	pulseglow = 0;
	OpMode = 1;

	EstablishSongStart = false;
	SongEnd = false;

	BothPressed = 0;
	DecPressed = 0;
	IncPressed = 0;
	TimeMode = 0;
		//     0      No time adjust
		//     1      Selecting quantity to change
		//     ... Back to 0
	ChangeTime = 0;

	DoublePress = 0;
	DecPress = 0;
	IncPress = 0;
	PressButtonBuf = 0;
}

void loop ()
{
//Calculating the current time:

	currenttime = millis();
	mil[1] = mil[0];
	mil[0] = currenttime;
	double d_mil = mil[0] - mil[1];

	T_sec_act += d_mil/1000;
	PressButtonBuf += (d_mil/1000);
	T_days = (int)(T_sec_act/3600)/24;

	T_hrs = T_sec_act/3600;
	while (T_hrs > 24)                                          // Keeps hours
under 24
		T_hrs -= 24;
	T_h_ones = (int)((int)T_hrs % 10);
	T_h_tens = (int)(((int)T_hrs/10) % 10);

	T_mins = T_sec_act/60;
	while (T_mins > 60)                                         // Keeps minutes
under 60
		T_mins -= 60;
	T_m_ones = (int)((int)T_mins % 10) ;
	T_m_tens = (int)(((int)T_mins/10) % 10);

	T_secs = T_sec_act;
```

```
       while (T_secs > 60)                                       // Keeps seconds
under 60
              T_secs -= 60;
       T_s_ones = (int)((int)T_secs % 10);
       T_s_tens = (int)(((int)T_secs/10) % 10);

       if (T_s_ones != T_0)
       {
              SecondPassed = true;
       }
       else
              SecondPassed = false;
       T_0 = T_s_ones;

       T_msec = T_sec_act*100;

       while (T_msec > 1000)
              T_msec -= 1000;

       currentms += d_mil;

       if (currentms >= 200)
       {
              MSPassed = true;
              currentms = 0;
       }
       else
              MSPassed = false;

//Switching to OpMode 2 if at the top of the hour
/*
       if ((T_s_ones == 0) && (T_s_tens == 0) && (T_m_ones == 0) && (T_m_tens == 0))
       {
              OpMode = 2;
       }
*/
//----------[ Definition of Button Presses ]-------------

       bool dec = digitalRead(decPin);
       bool inc = digitalRead(incPin);

// Defining double press
       if ((dec == 1) && (inc == 1))
              BothPressed += 1;
       else
              BothPressed = 0;
       if (BothPressed >= 10)
       {
              DoublePress = 1;
              DecPress = 0;
              IncPress = 0;
              BothPressed = 0;
       }
// Defining decrement press
       if ((dec == 1) && (inc == 0))
              DecPressed += 1;
       else
              DecPressed = 0;
       if (DecPressed >= 10)
       {
              DoublePress = 0;
              DecPress = 1;
              IncPress = 0;
              DecPressed = 0;
       }
// Defining increment press
       if ((dec == 0) && (inc == 1))
              IncPressed += 1;
       else
              IncPressed = 0;
       if (IncPressed >= 10)
       {
              DoublePress = 0;
              DecPress = 0;
              IncPress = 1;
              IncPressed = 0;
       }

//Serial.print("Dec | "); Serial.print(DecPress); Serial.print(" | Inc | ");
Serial.print(IncPress); Serial.print(" | Both | "); Serial.print(DoublePress);
Serial.print(" | ");
Serial.print(DecPressed);Serial.print(IncPressed);Serial.println(BothPressed);
//Serial.print("Dcp | "); Serial.print(DecPressed); Serial.print(" | Icp | ");
Serial.print(IncPressed); Serial.print(" | Botp | "); Serial.println(BothPressed);

// ----------[ Time Change Mode ]-----------

if (ChangeTime == 0)
{
       digitalWrite(statusPin, 1);
       if ((TimeMode == 0) && (DoublePress == 1) && (PressButtonBuf >= timebuf))
       {
              TimeMode = 1;                    // Start changing the time
              DoublePress = 0;
       }
       else if ((TimeMode == 1))         // If in change time mode,
       {
              if ((IncPress == 1) && (PressButtonBuf >= timebuf))
       // and incremented, go to Minutes
              {
                     TimeMode = 2;
                     IncPress = 0;
                     PressButtonBuf = 0;
              }
              if ((DecPress == 1) && (PressButtonBuf >= timebuf))
       // and decremented, go to Hours
              {
                     TimeMode = 3;
                     DecPress = 0;
                     PressButtonBuf = 0;
              }
              if ((DoublePress == 1) && (PressButtonBuf >= timebuf))
              {
                     ChangeTime = 1;
                     PressButtonBuf = 0;
                     DoublePress = 0;
                     digitalWrite(statusPin, 0);
              }
       }
       else if ((TimeMode == 2))         // If in change time mode,
       {
              if ((IncPress == 1) && (PressButtonBuf >= timebuf))
       // and incremented, go to Minutes
              {
                     TimeMode = 3;
                     IncPress = 0;
```

```
                        PressButtonBuf = 0;
                }
                if ((DecPress == 1) && (PressButtonBuf >= timebuf))
        // and decremented, go to Hours
                {
                        TimeMode = 1;
                        DecPress = 0;
                        PressButtonBuf = 0;
                }
                if ((DoublePress == 1) && (PressButtonBuf >= timebuf))
                {
                        ChangeTime = 1;
                        PressButtonBuf = 0;
                        DoublePress = 0;
                        digitalWrite(statusPin, 0);
                }
        }
        else if ((TimeMode == 3))         // If in change time mode,
        {
                if ((IncPress == 1) && (PressButtonBuf >= timebuf))
        // and incremented, go to Minutes
                {
                        TimeMode = 1;
                        IncPress = 0;
                        PressButtonBuf = 0;
                }
                if ((DecPress == 1) && (PressButtonBuf >= timebuf))
        // and decremented, go to Hours
                {
                        TimeMode = 2;
                        DecPress = 0;
                        PressButtonBuf = 0;
                }
                if ((DoublePress == 1) && (PressButtonBuf >= timebuf))
                {
                        ChangeTime = 1;
                        PressButtonBuf = 0;
                        DoublePress = 0;
                        digitalWrite(statusPin, 0);
                }
        }

}
else if (ChangeTime == 1)
{
        if ((TimeMode == 1))
        {
                if ((IncPress == 1) && (PressButtonBuf >= timebuf))
        // and incremented, go to Minutes
                {
                        T_sec_act += 1;
                        PressButtonBuf = 0;
                }
                else if ((DecPress == 1) && (PressButtonBuf >= timebuf))
        // and decremented, go to Hours
                {
                        T_sec_act -= 1;
                        PressButtonBuf = 0;
                }
                else if ((DoublePress == 1) && (PressButtonBuf >= timebuf))
                {
                        ChangeTime = 0;
                        PressButtonBuf = 0;
```

```
			DoublePress = 0;
		}
		if ((T_s_ones == 1))
		{
			digitalWrite(statusPin, 1);
		}
		else
		{
			digitalWrite(statusPin, 0);
		}
	}
	else if ((TimeMode == 2))
	{
		if ((IncPress == 1) && (PressButtonBuf >= timebuf))
	// and incremented, go to Minutes
		{
			T_sec_act += 60;
			PressButtonBuf = 0;
		}
		else if ((DecPress == 1) && (PressButtonBuf >= timebuf))
	// and decremented, go to Hours
		{
			T_sec_act -= 60;
			PressButtonBuf = 0;
		}
		else if ((DoublePress == 1) && (PressButtonBuf >= timebuf))
		{
			ChangeTime = 0;
			PressButtonBuf = 0;
			DoublePress = 0;
		}
		if ((T_s_ones == 1) || (T_s_ones == 3))
		{
			digitalWrite(statusPin, 1);
		}
		else
		{
			digitalWrite(statusPin, 0);
		}
	}
	else if ((TimeMode == 3))
	{
		if ((IncPress == 1) && (PressButtonBuf >= timebuf))
	// and incremented, go to Minutes
		{
			T_sec_act += 3600;
			PressButtonBuf = 0;
		}
		else if ((DecPress == 1) && (PressButtonBuf >= timebuf))
	// and decremented, go to Hours
		{
			T_sec_act -= 3600;
			PressButtonBuf = 0;
		}
		else if ((DoublePress == 1) && (PressButtonBuf >= timebuf))
		{
			ChangeTime = 0;
			PressButtonBuf = 0;
			DoublePress = 0;
		}
		if ((T_s_ones == 1) || (T_s_ones == 3) || (T_s_ones == 5))
		{
			digitalWrite(statusPin, 1);
			}
			else
			{
				digitalWrite(statusPin, 0);
			}
		}
}




IncPress = 0;
DecPress = 0;
DoublePress = 0;


//If clock mode is currently active

	if (OpMode == 1)
	{
		DivNumber = 0;

		dataS = GetBinary_ByteForm(T_h_ones, T_h_tens);
		dataM = GetBinary_ByteForm(T_m_ones, T_m_tens);
		dataH = GetBinary_ByteForm(T_s_ones, T_s_tens);

		for (int j = 0; j < 1; j++)
		{
			digitalWrite(latchPin, 0);			// Start listening
			shiftOut(dataPin, clockPin, dataH);
			shiftOut(dataPin, clockPin, dataM);
			shiftOut(dataPin, clockPin, dataS);
			digitalWrite(latchPin, 1);			// Stop listening
		}

		pulseglow = fabs( 255 * ( sin( (3.1415)*(((double)T_msec)/200) ) ) );
		// NOTE: The function above creates a sinusoidal PWM with an amplitude of
255, period of 2 seconds,
		// and the absolute value gives it the "bouncing" effect.
		ledcWrite(1,pulseglow);

		if (SecondPassed == true)
		{
			Serial.print(T_h_tens);
			Serial.print(T_h_ones);
			Serial.print(":");
			Serial.print(T_m_tens);
			Serial.print(T_m_ones);
			Serial.print(":");
			Serial.print(T_s_tens);
			Serial.println(T_s_ones);
		}
	}

	//if (T_s_ones <= 5)
	//	digitalWrite(33,HIGH);
	//else if (T_s_ones > 5)
	//	digitalWrite(33,LOW);

//If divergence mode is currently active (at the top of the hour)

	if (OpMode == 2)
	{
		if (EstablishSongStart == false)
		{
			Serial.println("Song Started.");
			SongStart = currenttime;
			EstablishSongStart = true;
			NextNote = 0;
			SongTime = 0;
			ActiveNote = 0;
			digitalWrite(divPin, 1);
		}
		SongTime = currenttime - SongStart - 1000;				// Get
current song time
		if (SongTime > SongStart+GOS_duration[ActiveNote])		// If it's
time to play the note,
		{
			ledcWriteTone(2,GOS_highnote[ActiveNote]);			//
write the note HZ value to the channel,
			ledcWrite(2,255);
	// then write to the buzzer pin.
			ActiveNote += 1;
	// Increment to the next note
		}


		if (ActiveNote > 14)
	// If we're at note fourteen
		{
			SongEnd = true;
		// the song is over
			Serial.println("Song ended.");
		}
		if (SongEnd == true)
	// and if the song is over,
		{
			EstablishSongStart = false;
	// let it play again
			SongEnd = false;
	// and make sure the song isn't over at the start
			OpMode = 1;
		// Go back to the clock
			ActiveNote = 0;
			NextNote = 0;
			ledcWrite(2,0);
		// Make sure sound is off
			digitalWrite(divPin, 0);
		}
		if (MSPassed == true)
		{
			dataH = GetBinary_ByteForm(DivMode_HT[DivNumber],
DivMode_HO[DivNumber]);
			dataM = GetBinary_ByteForm(DivMode_MT[DivNumber],
DivMode_MO[DivNumber]);
			dataS = GetBinary_ByteForm(DivMode_ST[DivNumber],
DivMode_SO[DivNumber]);

			for (int j = 0; j < 1; j++)
			{
				digitalWrite(latchPin, 0);			// Start
listening
				shiftOut(dataPin, clockPin, dataH);
				shiftOut(dataPin, clockPin, dataM);
				shiftOut(dataPin, clockPin, dataS);
```

```
                        digitalWrite(latchPin, 1);                     // Stop
listening
                    }


                    Serial.print(DivMode_HT[DivNumber]);
                    Serial.print(".");
                    Serial.print(DivMode_HO[DivNumber]);
                    Serial.print(DivMode_MT[DivNumber]);
                    Serial.print(DivMode_MO[DivNumber]);
                    Serial.print(DivMode_ST[DivNumber]);
                    Serial.println(DivMode_SO[DivNumber]);


                    if (DivNumber >= 19)
                          DivNumber = 19;
                    else
                          DivNumber += 1;
              }


       }
}

//=====================================================================================
===|
//===================================[ FUNCTIONS
]=======================================|
//=====================================================================================
===|


byte GetBinary_ByteForm(int Ones, int Tens)                                    //
Converts ones and tens place integers to 1 byte;
{
                    // binary form joining both nybbles together so
       byte RB = (byte) Ones;
       // we can pass a single byte into the 74HC595 for each
       byte LB = (byte) Tens;
       // of the hours, minutes, and seconds place
       byte ByteForm = RB + (LB<<4);
       return ByteForm;
}
void DisplayNumbers()
{

}

void shiftOut(int myDataPin, int myClockPin, byte myDataOut) {
  // This shifts 8 bits out MSB first,
  //on the rising edge of the clock,
  //clock idles low

  //internal function setup
  int i=0;
  int pinState;
  pinMode(myClockPin, OUTPUT);
  pinMode(myDataPin, OUTPUT);

  //clear everything out just in case to
  //prepare shift register for bit shifting
  digitalWrite(myDataPin, 0);
  digitalWrite(myClockPin, 0);
  //for each bit in the byte myDataOut
  //NOTICE THAT WE ARE COUNTING DOWN in our for loop
  //This means that %00000001 or "1" will go through such
  //that it will be pin Q0 that lights.
  for (i=7; i>=0; i--)  {
    digitalWrite(myClockPin, 0);

    //if the value passed to myDataOut and a bitmask result
    // true then... so if we are at i=6 and our value is
    // %11010100 it would the code compares it to %01000000
    // and proceeds to set pinState to 1.
    if ( myDataOut & (1<<i) ) {
      pinState= 1;
    }
    else {
      pinState= 0;
    }

    //Sets the pin to HIGH or LOW depending on pinState
    digitalWrite(myDataPin, pinState);
    //register shifts bits on upstroke of clock pin
    digitalWrite(myClockPin, 1);
    //zero the data pin after shift to prevent bleed through
    digitalWrite(myDataPin, 0);
  }

  //stop shifting
  digitalWrite(myClockPin, 0);
}

//=====================================================================================
===|
//==================================[ RESOURCES
]======================================|
//=====================================================================================
===|
/*
 * Tom Titor (4Chan /a/ board) website and schematic:
 *           http://www.mindspring.com/~tomtitor/index.html
 *
 * Adafruit ESP32 HUZZAH32 Pinout diagram:
 *           https://cdn-learn.adafruit.com/downloads/pdf/adafruit-huzzah32-esp32-
feather.pdf?timestamp=1579493965
 *
 * Autononymous main page (further resources from me):
 *           https://autononymous.github.io/index.html
 *
 * Music note frequencies (to compose your own songs):
 *           https://pages.mtu.edu/~suits/notefreqs.html
 *
 */


// El Psy Kongroo.
```

Appendix E: Note-To-Frequency Lookup Table.

Note	Freq [Hz]	Wave [cm]	Note	Freq [Hz]	Wave [cm]	Note	Freq [Hz]	Wave [cm]
C_0	16.35	2109.89	C_3	130.81	263.74	C_6	1046.50	32.97
$C^{\#}_0/D^{b}_0$	17.32	1991.47	$C^{\#}_3/D^{b}_3$	138.59	248.93	$C^{\#}_6/D^{b}_6$	1108.73	31.12
D_0	18.35	1879.69	D_3	146.83	234.96	D_6	1174.66	29.37
$D^{\#}_0/E^{b}_0$	19.45	1774.20	$D^{\#}_3/E^{b}_3$	155.56	221.77	$D^{\#}_6/E^{b}_6$	1244.51	27.72
E_0	20.60	1674.62	E_3	164.81	209.33	E_6	1318.51	26.17
F_0	21.83	1580.63	F_3	174.61	197.58	F_6	1396.91	24.70
$F^{\#}_0/G^{b}_0$	23.12	1491.91	$F^{\#}_3/G^{b}_3$	185.00	186.49	$F^{\#}_6/G^{b}_6$	1479.98	23.31
G_0	24.50	1408.18	G_3	196.00	176.02	G_6	1567.98	22.00
$G^{\#}_0/A^{b}_0$	25.96	1329.14	$G^{\#}_3/A^{b}_3$	207.65	166.14	$G^{\#}_6/A^{b}_6$	1661.22	20.77
A_0	27.50	1254.55	A_3	220.00	156.82	A_6	1760.00	19.60
$A^{\#}_0/B^{b}_0$	29.14	1184.13	$A^{\#}_3/B^{b}_3$	233.08	148.02	$A^{\#}_6/B^{b}_6$	1864.66	18.50
B_0	30.87	1117.67	B_3	246.94	139.71	B_6	1975.53	17.46
C_1	32.70	1054.94	C_4	261.63	131.87	C_7	2093.00	16.48
$C^{\#}_1/D^{b}_1$	34.65	995.73	$C^{\#}_4/D^{b}_4$	277.18	124.47	$C^{\#}_7/D^{b}_7$	2217.46	15.56
D_1	36.71	939.85	D_4	293.66	117.48	D_7	2349.32	14.69
$D^{\#}_1/E^{b}_1$	38.89	887.10	$D^{\#}_4/E^{b}_4$	311.13	110.89	$D^{\#}_7/E^{b}_7$	2489.02	13.86
E_1	41.20	837.31	E_4	329.63	104.66	E_7	2637.02	13.08
F_1	43.65	790.31	F_4	349.23	98.79	F_7	2793.83	12.35
$F^{\#}_1/G^{b}_1$	46.25	745.96	$F^{\#}_4/G^{b}_4$	369.99	93.24	$F^{\#}_7/G^{b}_7$	2959.96	11.66
G_1	49.00	704.09	G_4	392.00	88.01	G_7	3135.96	11.00
$G^{\#}_1/A^{b}_1$	51.91	664.57	$G^{\#}_4/A^{b}_4$	415.30	83.07	$G^{\#}_7/A^{b}_7$	3322.44	10.38
A_1	55.00	627.27	A_4	440.00	78.41	A_7	3520.00	9.80
$A^{\#}_1/B^{b}_1$	58.27	592.07	$A^{\#}_4/B^{b}_4$	466.16	74.01	$A^{\#}_7/B^{b}_7$	3729.31	9.25
B_1	61.74	558.84	B_4	493.88	69.85	B_7	3951.07	8.73
C_2	65.41	527.47	C_5	523.25	65.93	C_8	4186.01	8.24
$C^{\#}_2/D^{b}_2$	69.30	497.87	$C^{\#}_5/D^{b}_5$	554.37	62.23	$C^{\#}_8/D^{b}_8$	4434.92	7.78
D_2	73.42	469.92	D_5	587.33	58.74	D_8	4698.63	7.34
$D^{\#}_2/E^{b}_2$	77.78	443.55	$D^{\#}_5/E^{b}_5$	622.25	55.44	$D^{\#}_8/E^{b}_8$	4978.03	6.93
E_2	82.41	418.65	E_5	659.25	52.33	E_8	5274.04	6.54
F_2	87.31	395.16	F_5	698.46	49.39	F_8	5587.65	6.17
$F^{\#}_2/G^{b}_2$	92.50	372.98	$F^{\#}_5/G^{b}_5$	739.99	46.62	$F^{\#}_8/G^{b}_8$	5919.91	5.83
G_2	98.00	352.04	G_5	783.99	44.01	G_8	6271.93	5.50
$G^{\#}_2/A^{b}_2$	103.83	332.29	$G^{\#}_5/A^{b}_5$	830.61	41.54	$G^{\#}_8/A^{b}_8$	6644.88	5.19
A_2	110.00	313.64	A_5	880.00	39.20	A_8	7040.00	4.90
$A^{\#}_2/B^{b}_2$	116.54	296.03	$A^{\#}_5/B^{b}_5$	932.33	37.00	$A^{\#}_8/B^{b}_8$	7458.62	4.63
B_2	123.47	279.42	B_5	987.77	34.93	B_8	7902.13	4.37

Table adapted from pages-mtu.edu [8].

Appendix F: Legal Disclaimer

DISCLAIMER

All material covered in this document, attached files, and included print assets is for informational purposes only. I take no responsibility for what you do with this knowledge. I can not be held responsible for any property or medical damages caused by the items described in this document. I advise you to check local laws and consult professional electricians/contractors for any project involving electricity, construction or assembly.

The DIY and tutorial material taught throughout this publication and attached materials are solely for informational purposes. By taking any information or educational material from this publication and attached materials, you assume any and all risks for the material covered. You agree to indemnify, hold harmless, and defend Ryan M. Kissinger from any and all claims and damages as a result of any and all of the information covered.

By taking and/or using any informational resources from Ryan M. Kissinger, you agree that you will use the information in this document in a safe and legal manner, consistent with any and all applicable laws, safety rules, and common sense. You further agree that you will take such steps as may be reasonably necessary or required by law to keep any information out of the hands of minors and immature and/or unqualified individuals.

You must accept that you and you alone are accountable for your safety, as well as the safety of others in any endeavor. While the material in this document and attached sources is provided in hopes that you construct your own project, you are ultimately responsible for verifying its applicability and accuracy to your project. You are completely responsible for knowing your limitations of knowledge and experience. If you do any work with high voltage power such as 120 or 240 VAC power wiring, you should consult a Licensed Electrician.

Some illustrations do not depict safety precautions or necessary equipment, in order to show the project steps as clearly as possible. These projects are not intended for use by individuals under the age of 18. Use of these instructions and suggestions is at your own risk. Ryan M. Kissinger disclaims all responsibility for any resulting damage, injury, or expense. It is your responsibility to make sure that your activities comply with any and all applicable laws.

By proceeding in the manufacturing or assembly of this project, you agree that you have read and understood this Disclaimer.

Works Cited

[1] J. Boos, "The Nixie tube story," in *IEEE Spectrum*, vol. 55, no. 7, pp. 36-41, July 2018.

[2] G. Zorpette, "New life for Nixies [in digital clocks]," in *IEEE Spectrum*, vol. 39, no. 6, pp. 44-49, June 2002.

[3] *74HC595 Product Datasheet.* Nexperia B.V., 2017, p. 1-5, 16-18 [Online]. Available: https://assets.nexperia.com/documents/data-sheet/74HC_HCT595.pdf. [Accessed: 12-February-2020]

[4] *National TTL Databook.* National Semiconductor Corp., 1976, p. 1-4 [Online]. Available: https://datasheetspdf.com/parts/DM5441A.pdf?id=866041. [Accessed: 12-February-2020]

[5] *ESP32 Series Datasheet.* EspressIf Systems, 2019, p 1-4 [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. [Accessed: 12-February-2020]

[6] *Adafruit HUZZAH32 ESP32 Feather.* Adafruit, 2019, p1-4 [Online]. Available: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-huzzah32-esp32-feather.pdf?timestamp=1581553421. [Accessed: 12-February-2020]

[7] Zeyuan. Yan, *NCH6100HV Datasheet.* p 1-4 [Online]. Available: https://elty.pl/pl/p/file/ca247d15f86f21efcaf90591a05a01f2/NIXIE-Power-Supply-Datasheet-EN-v1.0.0.pdf. [Accessed: 13-February-2020]

[8] B. H. Suits, *Physics of Music - Notes*, 1998. [Online]. Available: https://pages.mtu.edu/~suits/notefreqs.html. [Accessed: 19-Feb-2020].

[9] C. Maw and T. Igoe, *Arduino ShiftOut Reference*, 25-Oct-2006. [Online]. Available: https://www.arduino.cc/reference/en/language/functions/advanced-io/shiftout/. [Accessed: 19-Feb-2020].