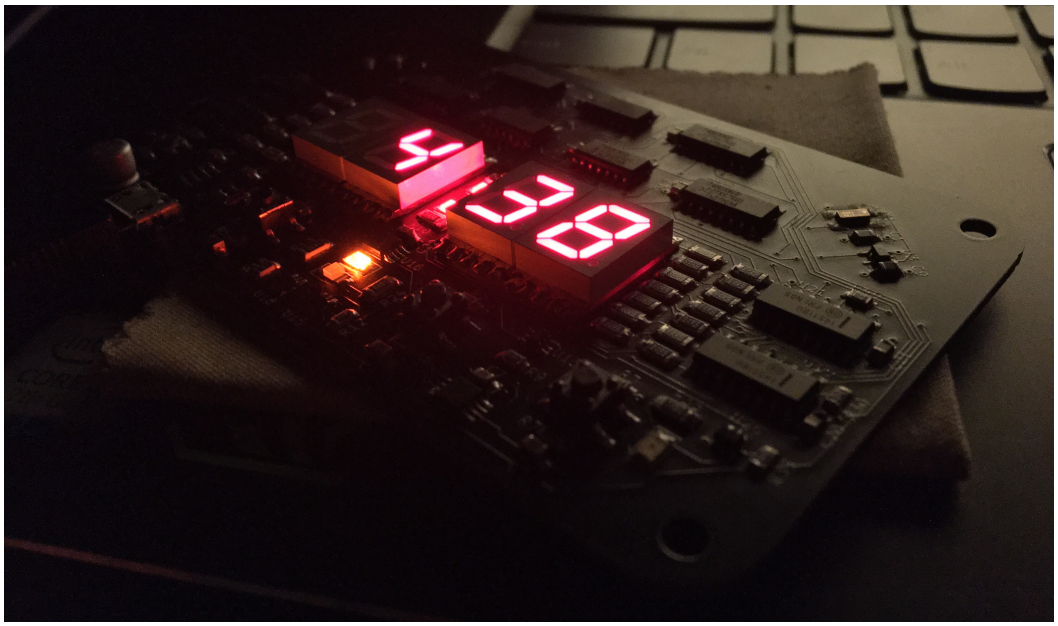
12-Hour Discrete Logic CMOS Clock

RM



Contents

1 Description

The general principle of this board is simple: take a 1Hz clock signal and feed it through asynchronous logic to display the time in a 12-hour format, with a seconds blinker and AM/PM indicator. The main motivation behind this project was to expand my PCB design skills with a not so complex design, coupled with my interest in digital logic. I figured that making a digital clock with 7400 series CMOS logic was a neat idea, and also because I am a fan of seven-segment displays (my GitHub display picture is a dead giveaway). There are many well made designs for digital logic clocks on the web, but I wanted to take an already existing design and modify it a little bit. This board is specifically a derivative of Erik van Zijst's design, so I need to give him credit for the idea (you can view his Medium write-up via this link).

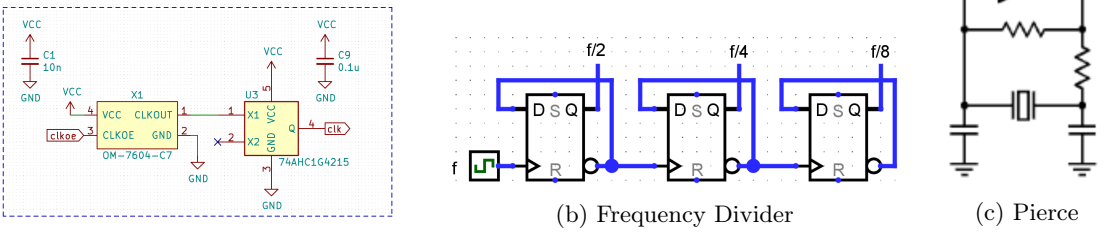
Overall, the entirety of the project timeline consisted of:

- Circuit design and prototyping
- Part selection and research (lots of datasheet reading)
- Schematic capture and custom symbol/footprint generation
- PCB layout/routing and modification

2 Implementation

All if not most of this design is modular. The main aspects are asynchronous sequential logic for output manipulation and interfacing, LED driving, and PWM (pulse width modualtion) for brightness control. Normally, a breadboard would be used for prototyping, but I decided to use Logisim Evolution to speed up the design process.

2.1 1Hz Clock Signal



(a) Schematic

(b) Frequency Divider

(c) Pierce Oscillator (Maxim Integrated)

Figure 1: Clock Signal Components

Probably the most important part of the design is needing a precise 1 Hz clock signal to drive the cascaded counter logic used to display the hours and minutes. This is usually done via a frequency divider and quartz crystal oscillator with a resonant frequency that is a power of 2 (32768 Hz typically). A frequency that is a power of 2 is specifically needed due to the way the frequency divider works, which is simply a chain of flip-flops, more commonly known as an asynchronous ripple counter (Figure 1 (b)). The counter exploits the memory element of a flip-flop by tying the inverting output back to the input. At every clock edge (positive or negative depending on the flip-flop architecture), the data input D is output to Q. The inverting output $\overline{\text{Q}}$ then negates this bit value at input D, and at the next clock edge the opposite value will be output. The resulting output frequency is halved and cascading a chain of these will divide the initial input frequency by 2^N, where N is the number of flip-flops. For a crystal which is tuned to a frequency of 32768 Hz, 15 flip-flops are needed.

The input frequency can be generated via a pierce oscillator configuration (Figure 1 (c)), which uses a quartz crystal combined with load capacitors, feedback and series resistors, and a digital inverter. This is a lengthy topic to explain, but generally speaking, quartz crystals exhibit an inverse piezoelectric effect whereby applying a voltage across it will deform it slightly. When this voltage is removed, the crystal resonator generates a voltage at a its resonant frequency that can then be amplified and fed back to it to sustain oscillation. There is a lot of theory behind this and it is not the focus of my documentation, but this PDF provides a detailed explanation on how it works.

If you look at the datasheet for any quartz crystal, a load capacitance C_L is specified. The corresponding load capacitors C_1 and C_2 that generate the necessary phase shift and frequency adjustment for the crystal are then calculated via the equation $C_1 = C_2 = 2(C_L + C_{stray})$, where C_{stray} is the stray capacitance from the PCB traces and integrated circuit inputs (MOSFET gates). C_{stray} is usually estimated to be around 3 to 5 picofarads. The series resistor provides additional phase shift and output drive isolation to protect the crystal, and is usually a few kilo Ohms. The feedback resistor linearizes the digital inverter to act as an amplifier is usually integrated within the IC. Instead of using a pierce oscillator, I opted to use an external 32768 Hz CMOS crystal oscillator module for one reason: less risk. This would guarantee an output frequency that I needed, and I would not need to worry about getting the load capacitance values wrong if I used a pierce oscillator instead (i.e. the stray capacitance might have been different from the estimated

value). It also provides better stability and precision, along with an enable pin that I actually used later on in the design for synchronization with the actual time.

As I said before, you would need a chain of 15 flip-flops to divide a 32768 Hz clock. However, before 2019, there existed no such IC. Instead, this was done via a 14 stage divider coupled with a single output flip-flop. Fortunately, Nexperia released a 15 stage divider in 2019, so I was able to generate a precise 1 Hz clock signal with only 2 ICs.

2.2 Clock Counter Logic

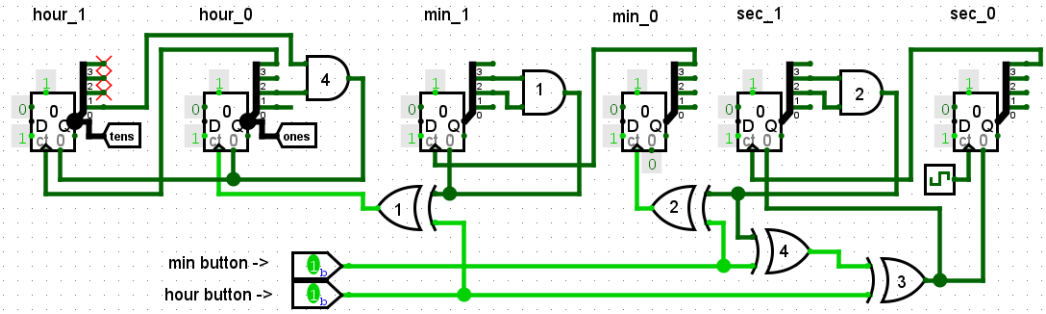


Figure 2: Timekeeping and Asynchronous Override Logic

The clock counter logic is what drives the output display time, namely, the minutes, and hours. In Logisim, a counter can be configured to however many bits are needed. Since we need to go up to a max value of 9 for the ones place digit, a 4-bit counter must be used. Figure 2 consists of the 4-bit time counters, AND gates that reset the counters, and XOR gates that interface with the manual push button signals to override the internal clock.

The logic can be more easily understood by following the diagram from right to left. The 1 Hz clock signal is sent to the first counter on the right (sec_0 in Figure 2), which controls the ones digit for the seconds. A 4-bit counter can count up to a max value of 15, however we can configure it to have a max value of 9. This is usually referred to as a decade counter (74HC390). The 74HC390 specifically is triggered by a negative edge on the clock input, which can be used to our advantage for the subsequent counters. To trigger the tens digit (sec_1), we can simply tie the MSB (most significant bit) of sec_0 to its clock input, since it outputs a negative edge at the reset of 9 to 0 (1001 to 0000). The problem on sec_1 however is that it will continue until 9 and must reset at 5 for both counters to count up to 59 instead of 99. To fix this, we cant reset sec_0 every time it hits 6 (0110) by tying an AND gate (AND gate 2 in the diagram) to bits 1 and 2. In reality, it resets a few nanoseconds after hitting 60 but this delay is negligible. Furthermore, we can use this to drive the minute counter ones digit (min_0), which drives the tens digit (min_1) just like the seconds counters. If you look at the diagram, I don't actually tie the output of AND gate 2 to the reset of sec_1, but rather to XOR gates 4 and 2.

To easily interface the counter logic with the manual buttons, I wanted a simpler way that used less components rather than multiplexing with a switch for example. I came up with an 4 gate XOR configuration (one IC) that allows both the internal clocks and manual inputs to update the time asynchronously, while

at the same time resetting the second counters when a manual button was pressed. A reset every time a manual button is pressed would ensure that there would be no clock signal from the sec_1 when a minute has passed, and keeping it held would halt the seconds from increasing. XOR gates were specifically used due to the fact that a high to low or low to high transition can be output to both the clock and reset lines. Take for example XOR gate 4. I stated previously that I don't tie the reset line of sec_1 to itself. Rather, it goes through XOR gates 4 and 3. Because the manual inputs are inactive high, a trigger reset by AND gate 2 would set XOR gate 4 low, which consequently sets XOR gate 3 high, resetting both counters. Any push by the minutes or hours manual button, or a reset signal by AND gates 1 or 2, would drive XOR gates 1 or 2 high to low, increasing the value on the respective minute or hour counter.

The difference for the hours counter is that it does not go from 00 to 59 like seconds and minutes, but rather 1 to 12. The problem is that the counters are designed to start at 0, so we need a way to manipulate the output such that the ones digit for the hours display starts at 1. I came up with two ways to do this, which I discuss in the next section. Because of the way I manipulate the hours digits, I use AND gate 4 to reset the hours counters when 12 is reached. For example, a logic 1 on bit 0 for hour_1 and logic 1 on bit 2 for hour_0 is 12 in BCD (binary coded decimal) (0001 0010). This forces the hours counters to count from 00 to 11 only, but 1 to 12 is what is seen on the seven segment displays.

2.3 Output Manipulation

Implementing a 24-hour format is much easier than a 12-hour format because it allows the hours counters to start at 0. In a 12-hour format, the hours must start at 1, and the tens digit must be blanked when 10, 11, 12 are not displayed. To reiterate, we can force the hour counters to reset upon reaching 12 (00 to 11), and manipulate the logic such that a 12-hour format is displayed. I realized two ways of doing this; the first being to display 12 instead of 0 (zero detection and multiplexing), and second being to utilize a 4-bit adder to start at 1 rather than 0. I decided to go with the former as it made more sense from a logical perspective while also requiring only two logic types. The adder method on the other hand is slightly non-intuitive and uses more logic types.

2.3.1 Method 1: Zero Detection and Multiplexing

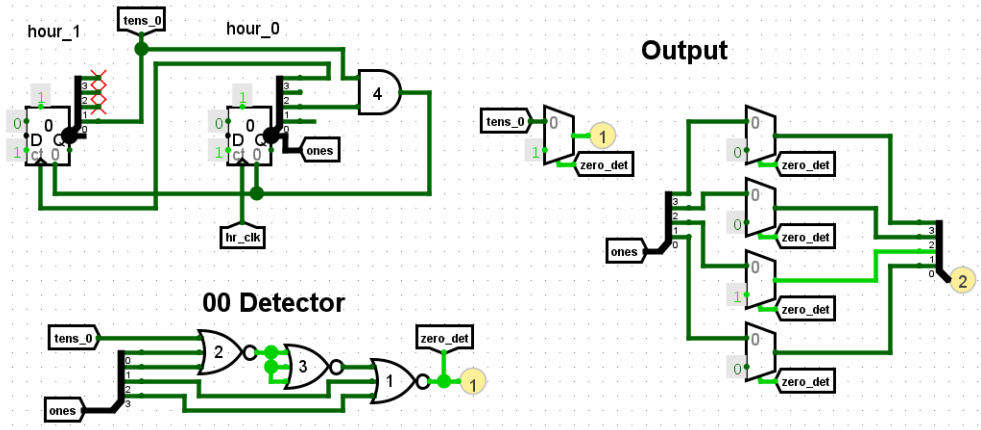


Figure 3: Multiplexing and NOR logic

To generalize the purpose of this section, I take the 4-bit lane from hour_0 and and bit 0 from hour_1 and attach them to labels. I could have easily connected the actual lines from the hours counters to the NOR gates and multiplexers, but separated each one to organize the diagram. The hour clock is unchanged, and it still arrives from XOR gate 1 in Figure 2, so what is shown in Figure 3 is what is needed (i.e. this section independent from the minutes and seconds logic).

Detecting a 00 on both hours counters and displaying a 12 can be thought of as saying if both counters are 00, display 12, otherwise display their actual value. In register transfer level (RTL), or describing the logic in Verilog, this section (purely combinational) would require the use of a multiplexer for if-else selection and a combinational circuit for 00 detection.

Detecting a 00 on both counters only needs a 5 input NOR gate, which outputs a logic high if all inputs are logic low, and logic low otherwise. You might think that you need all 8 bits of both counters, but for 12-hour format, only a 1 or blank is displayed on the tens digit which is why we only need bit 0 of hour_1. There are many ICs that implement a 5 input NOR gate, but out of all the ones I have seen, they contain two 5 input NOR gates. Technically, I could have used these chips and only used 1 gate, but since I like to optimize for utilization and space, I decided to use a three 3 input NOR gate IC (74HC27D) that can be configured to be equivalent to a single 5 input NOR gate. This would save me the space of using a 14 pin IC over a 16 pin IC. The output of NOR gate 1 is connected to the select bit of all multiplexers in the output section.

Once a 00 is detected, NOR gate 1 will output logic high and each multiplexer will then select the preconfigured logic level (decimal 12 or 1 0010 in BCD, excluding the unused bits from hour_1), which is either tied to VCC (5V) or ground. If 00 is not detected, the multiplexers take the counter value lines.

In order to blank the seven segment that displays hour_1 whenever it is a 0 (i.e. 01 should be 1), we can tie bit 0 of hour_1 straight to the inverting blanking input ($\overline{\text{BI}}$) of the respective BCD to seven segment decoder. This will blank the display so no number is displayed, but because I also wanted some form of brightness control, it is tied to the fourth AND gate (notice in Figure 2 I only use three AND gates) along with the 7555 PWM circuit output (section 2.4), where the output of that AND gate connects to the blanking input. This ensures that the display will only be modulated whenever a “1” appears on the display.

2.3.2 Method 2: 4-bit Adder

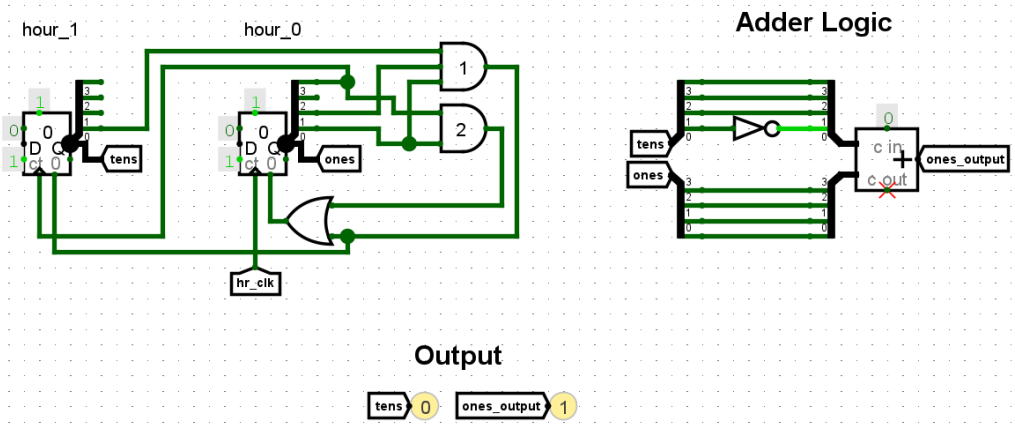


Figure 4: Adder and Reset Logic

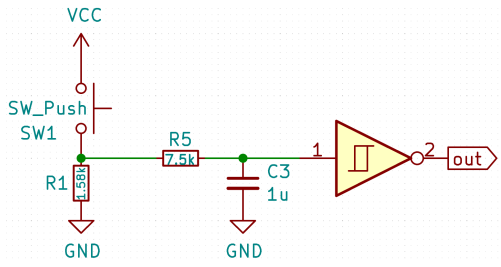
This method utilizes a 4-bit adder along with reset logic on the hour counters. Unlike the zero detection that has an initial state of 12, the initial state of this method is 01.

My first inclination was to tie hour_0 to the adder and add it with a constant 0001. This will work as the output for the this digit will at 1 rather than 0, but only for counter values 0 to 8, since the output will be 1010 (10) when hour_0 reaches a value of 1001 (9). A solution to this is to not use a constant bit value, but the complement of hour_0 bit 0. On the first iteration (hour_0 counting from 0 to 8) the output for the ones digit will be 1 to 9. Since we do not want a 10 at the output for the ones digit, we must reset hour_0 at 9, which is done by AND gate 2. When bit 3 of hour_0 provides a negative edge (a reset at 9) to the clock input of hour_1), hour_1 will be 0001. However, since the inverter is in place, the input to the adder sees a 0000, and the output is equivalent to hour_0, which is 0. This will display a 10 on the seven segments after a 09.

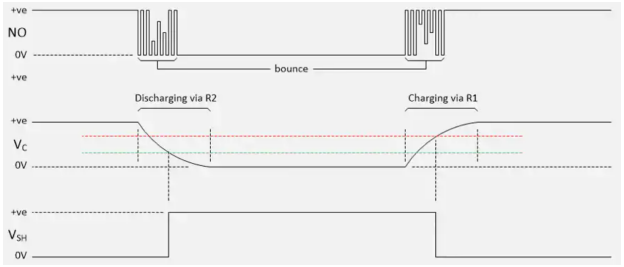
Once a 10 is displayed, the next reset for both counters needs to be at 13. To do this, AND gate 1 is used to detect a (0001) 1 and (0011) 3 on hour_1 and hour_0, respectively, which then connects to the reset of hour_0 through an OR gate and the reset of hour_1. Effectively, we have two resets; one when hour_0 hits a 9 and the other when hour_1 and hour_0 are BCD 13 (0001 0011), prompting the use of the OR gate for the two reset conditions on hour_0.

Comparing this strategy to method 1 yields more logic types, and therefore more ICs, which means more cost for my frugal attitude. This is why I chose to go with method 1 as it was simpler to implement and required only 3 ICs of 2 logic types (the single multiplexer in Figure 3 is a tiny TSOP6 package).

2.4 Manual Button Debouncing



(a) Schematic



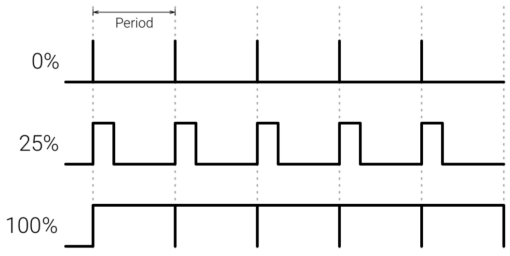
(b) Filtered and Buffered Waveforms (Digi-Key)

Figure 5: Debouncing Waveforms and Schematic

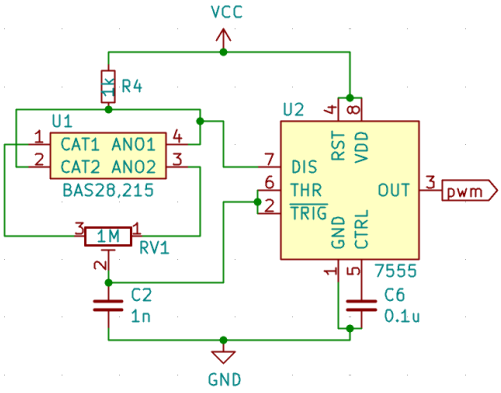
Ideally, pressing a push button who's terminals are connected to the supply voltage via a pull up or down resistor should give a waveform that does not contain intermediate switching between supply and ground, as seen in the bottom waveform in Figure 5b. Unfortunately, push buttons are not perfect and shift between contacts when they are pressed. To fix this, a switch debouncing circuit is commonly used that consists of a low pass RC filter and an inverting Schmitt trigger buffer, which filters the high frequency bouncing and buffers the filtered signal to output a high or low transition to a proceeding stage. Choosing the values of the resistors and capacitor is based on the desired difference in time from pressing or releasing the button to having an output signal, and the threshold voltage on the Schmitt trigger.

The 74AHC3G14 tripple inverting Schmitt trigger buffer was used since I use 3 buttons on the board (2 for time incrementing and 1 for halting the clock generator). The datasheet for the 74AHC3G14 specifies positive and negative going threshold voltages of 3.8V and 1.6V, respectively, which determine the transition point of the output signal. The first order differential solution to the RC charging circuit is given as $V_c = V_b(1 - e^{-t/RC})$, and can be used to set the RC values based on the specified time t, which is how long voltage V_c will take to reach a max voltage V_b with an initial voltage of 0V. On average, push buttons bounce for around 2ms, and t can be set to 10ms for a safe margin. To reach the positive going threshold voltage of 3.8V, I set the capacitor value to 1uF and solving for R yielded 7kΩ, which the most common value closest is 7.5k. Conversely, finding the value on the discharging portion for reaching the negative going threshold voltage of 1.6V from an initial value of 5V is solved via the equation $V_c = V_b(e^{-t/RC})$. Keeping the capacitor at a value of 1uF and time t at 10ms, R is given as 8.7k. The extra resistance of 1.58k gives a total discharging resistance of 9k, as seen in Figure 5a, which satisfies the time constant.

2.5 Pulse Width Modulation Brightness Control



(a) Various Duty Cycles (Analog Devices)



(b) Schematic

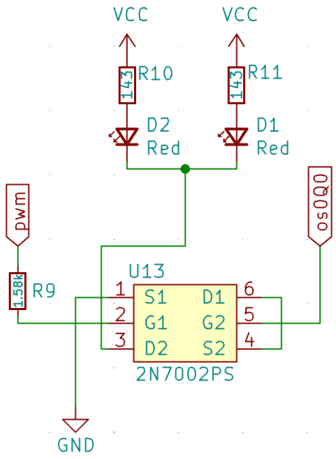
Figure 6: PWM Output Waveform and Generator

I chose to limit each LED, including the seven segment displays, to a maximum of 20mA. Even at 20mA, LEDs can get very bright, which is why a form of brightness control is needed. PWM is very useful that given a certain output duty cycle, the effective output power can be lowered proportional to the duty cycle. To get a tunable waveform like this, a 7555 timer with a modified astable configuration (addition of two diodes to control charge and discharge path) that can give a adjustable duty cycle at a fixed frequency is used. To learn more about how an astable configuration works, you can read this old report I made a while back. According to the 7555 datasheet, the duty cycle in the default astable configuration is given as $(R_A + R_B)/(R_A + 2R_B)$. Unfortunately, this equation relies on a fixed R_A and R_B without a modified circuit path. However, it shows that the ratio is directly proportional to the duty cycle, which provided that the ratio of R_A over R_B is very small or $R4$ over $RV1$ in this case (Figure 6b), a larger range can be produced. In fact, I tried this with a 1k resistor and 10k potentiometer I already had and found that the duty cycle could only reach around 70% maximum. Replacing the 10k potentiometer with 1M effectively gave a full range of 0 to 100% and by calculation, an output frequency of ∼1.4kHz, accounting for the charging capacitor and resistor tolerances.

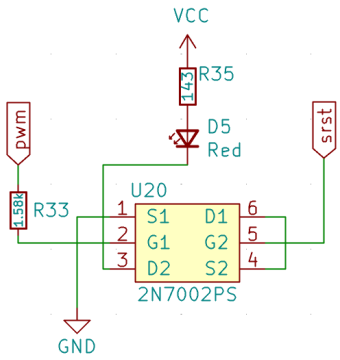
Furthermore, the output current capability of the 7555 is particularly poor at driving multiple loads at constant current. Although the datasheet states an absolute maximum output capability of 100mA, the output voltage drops severely when sourcing 10mA with a 5V supply. Due to this, the common method of driving constant current into 8 BJT LED drivers would be difficult and an NMOS solution would be necessary. Because I would be driving 8 CMOS inputs, I decided to add output resistors to limit the total output rush current to around 25mA when driving the output signal high. These resistors would also provide dampening to reduce ringing effects, and do not inhibit the turn on time of the CMOS loads at such a low frequency. As of writing this report, I think these resistors were unecessary, since the gate capacitance of the MOSFETs are very low. Still, the rise time of the resistor and gate capacitance product is a lot smaller than the PWM frequency of around 1.6kHz, so they do not inhibit the modulated square wave.

2.6 LED Drivers

2.6.1 Second Flashers and Reset LED



(a) Second Flashers



(b) Second Reset Indicator

Figure 7: NMOS NAND logic LED Drivers

To drive individual or parallel LEDs, I used a dual N-channel NMOS IC and connected the two in series (drain to source). The resulting logic is a NAND gate, since the output will only be low when both gate inputs (PWM and driving signal) are logic high, provided that there is a pull-up resistor to VCC/VDD. For the second flashers, the driving signal comes from bit 0 of the first counter. Although it was not necessary, I decided to add a second reset indicator every time a button was pushed, as a reassurance that a logic high was indeed sent to the reset pin of the first counter. This signal specifically comes from XOR gate 3 in Figure 2.

2.6.2 AM/PM Indicator

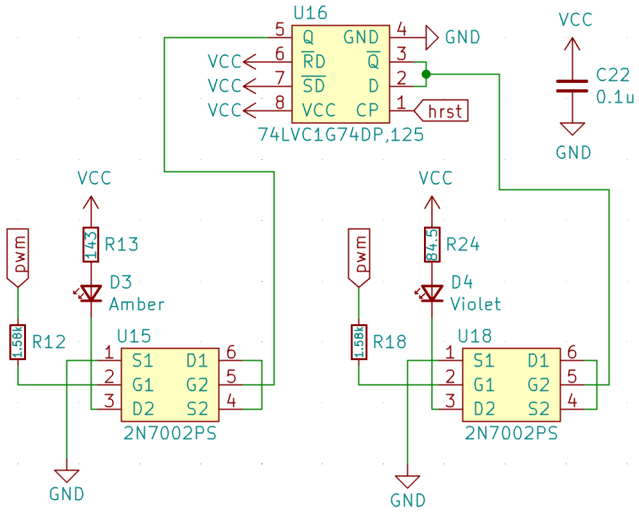


Figure 8: Divide by 2 Meridiem Flip-Flop and AM/PM LED drivers

An AM/PM indicator is without a doubt a must when using a 12-hour display format. The simple solution is to add a divide by 2 flip-flop to determine when the hours counters reset from 12 to 0 (i.e. when a 0001 and 0010 is detected on the counters, as seen in on AND gate 4 in figure 3). This reset signal serves as the positive edge clock input for the flip-flop, which can be used to drive the Q and $\overline{Q}$ outputs to show AM and PM. When Q is high, $\overline{Q}$ is low. The same applies vice versa, so a functional AM/PM indicator is displayed. Again, these outputs feed to dual NMOS in series with PWM so their output LED indicator brightness can be controlled.

2.6.3 BCD to Seven Segment Decoders/Seven Segment Displays

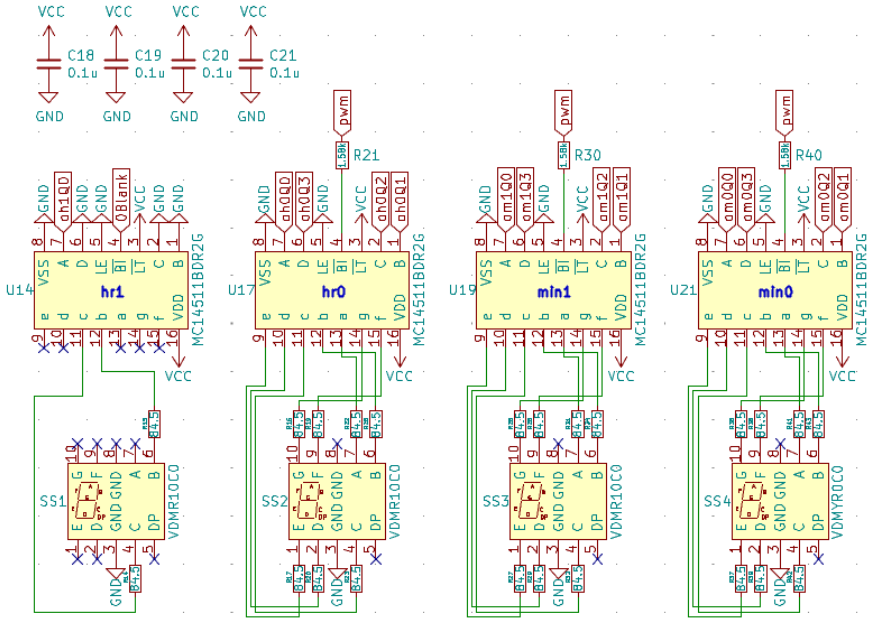


Figure 9: Seven segment decoders and LEDs

This section was straightforward as in connecting the outputs of the decade counters directly to seven segment decoders, and PWM to the blanking inputs. However, for a 12-hour display, it is criminal to display 0 on the hours tens digit when 11 or 12 are not displayed (i.e 01 should be just 1). Fortunately, I had one AND gate that was unused, so the output from the single 2 to 1 multiplexer (the hours tens digit - 0 or 1) in figure 3 and the PWM were tied to the inputs of it. The output was then connected to the blanking input, so a zero would be blanked as necessary, since the display is off whenever the blanking input is logic low (i.e. a 0 bit).

3 Physical Design

3.1 Schematic Capture

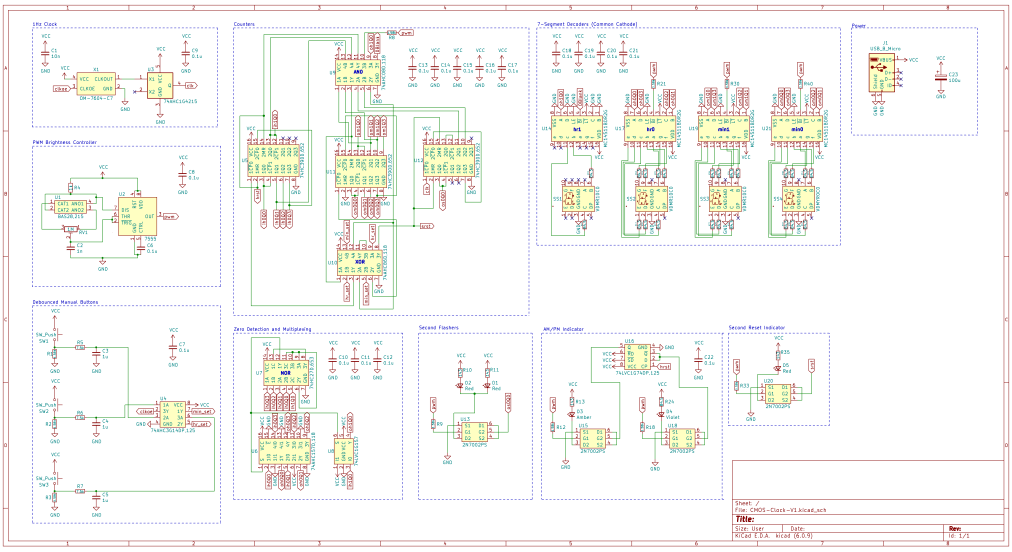


Figure 10: Full Schematic

The design was simple enough that I captured every symbol in one sheet. Additionally, I added a 150uF power supply bulk/filtering capacitor, although I think it was unnecessary at the time of writing this report. Since the total current consumption is around 500mA, I do not think a typical 5W adapter would have trouble supplying this current in bursts.

3.2 PCB

3.2.1 Initial Layout and Routing

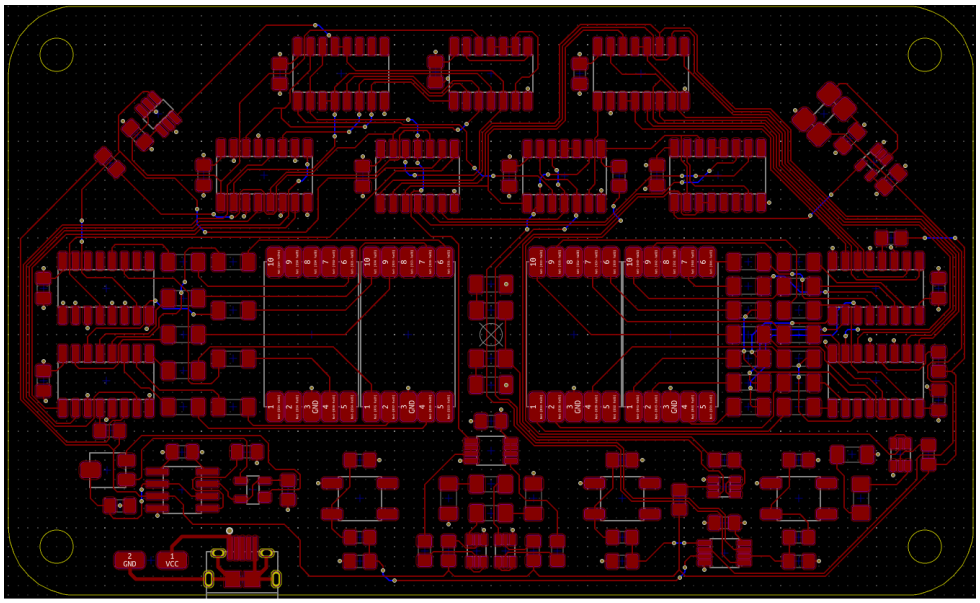


Figure 11: Initial Routing

The initial layout started with laying out the seven segment displays and decoders first. Because it is meant to be a visual display, I wanted it to look nice aesthetically, with the display in the middle, manual adjustment beneath, and logic on top. Figure 11 depicts my first attempt at routing, on a 4 layer setup (signal on top and bottom layers, power and ground on internal layers).

3.2.2 Design Considerations and Improvements

In terms of improvements and issues with the layout, I realized that I spent too much time utilizing the top layer for most of the routing. The issue that arose from this was not only wasting an entire bottom layer of copper to allow for easier routing, but because the traces on the top layer were tightly packed together. During this point, I learned of crosstalk, where the rise time in a signal trace, or current flow, can generate a magnetic field and create unwanted coupling to another trace, changing its voltage. Because I also used advance high speed CMOS chips, the rise times in these ICs are very short (a few nanoseconds), and they could have severely affected the clock traces to each of the decade counters, had I not been careful. As such, I calculated the coupled voltage through Saturn PCB design, and added spacing to the traces that would be most affected. This also forced me to move some traces to the bottom layer. Figure 12 below depicts the finalized layout which I used.

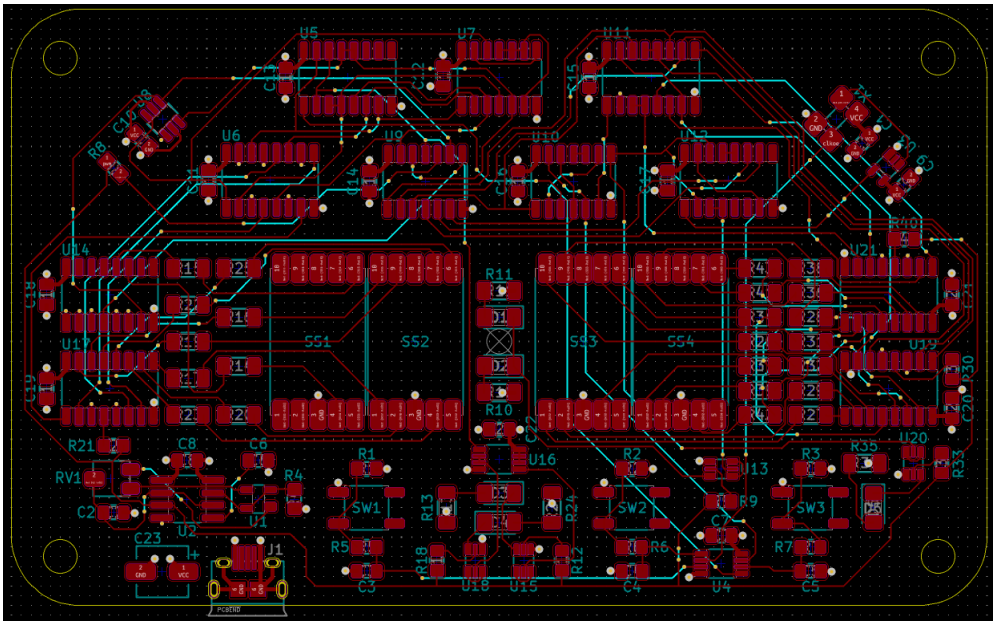


Figure 12: Improved Layout

3.3 Result

I ordered the PCB for manufacturing by JLCPCB. The figures below show the finished board.

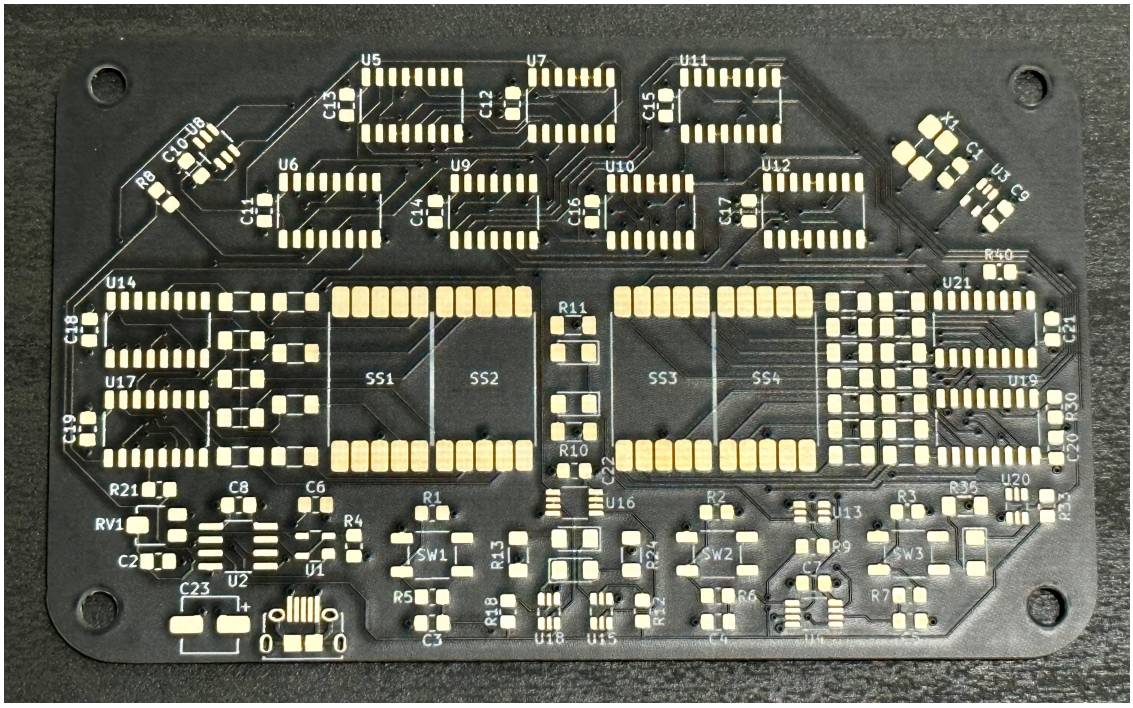


Figure 13: Front Side of PCB

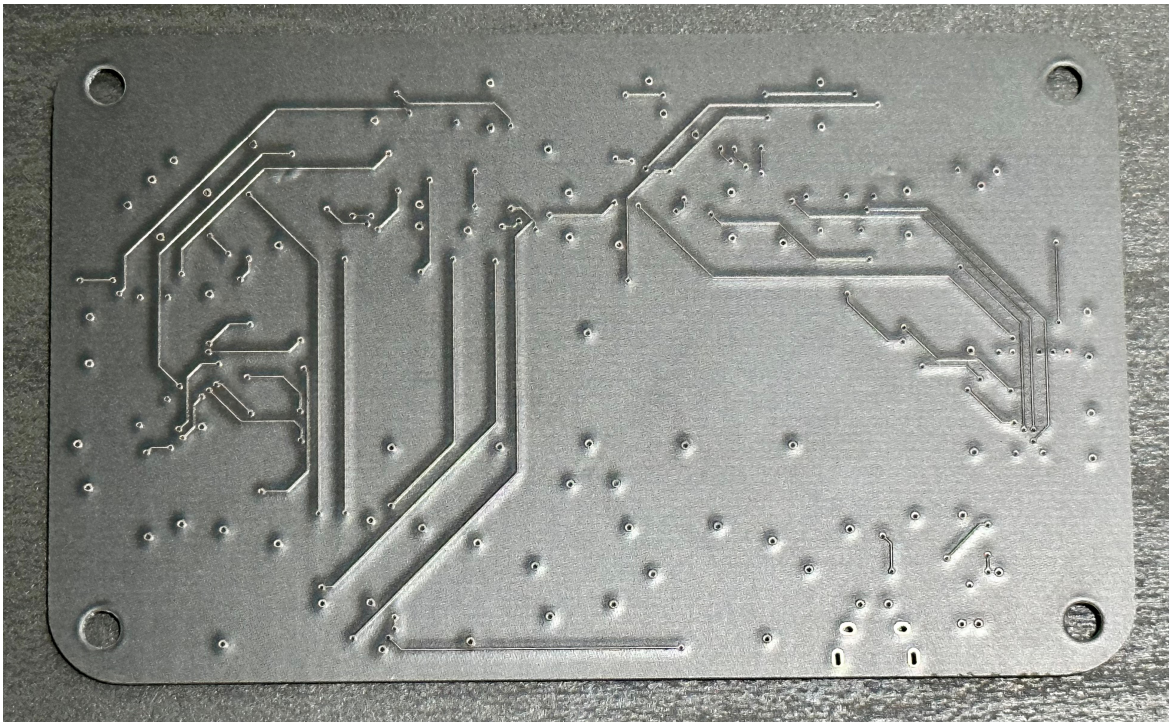


Figure 14: Back Side of PCB

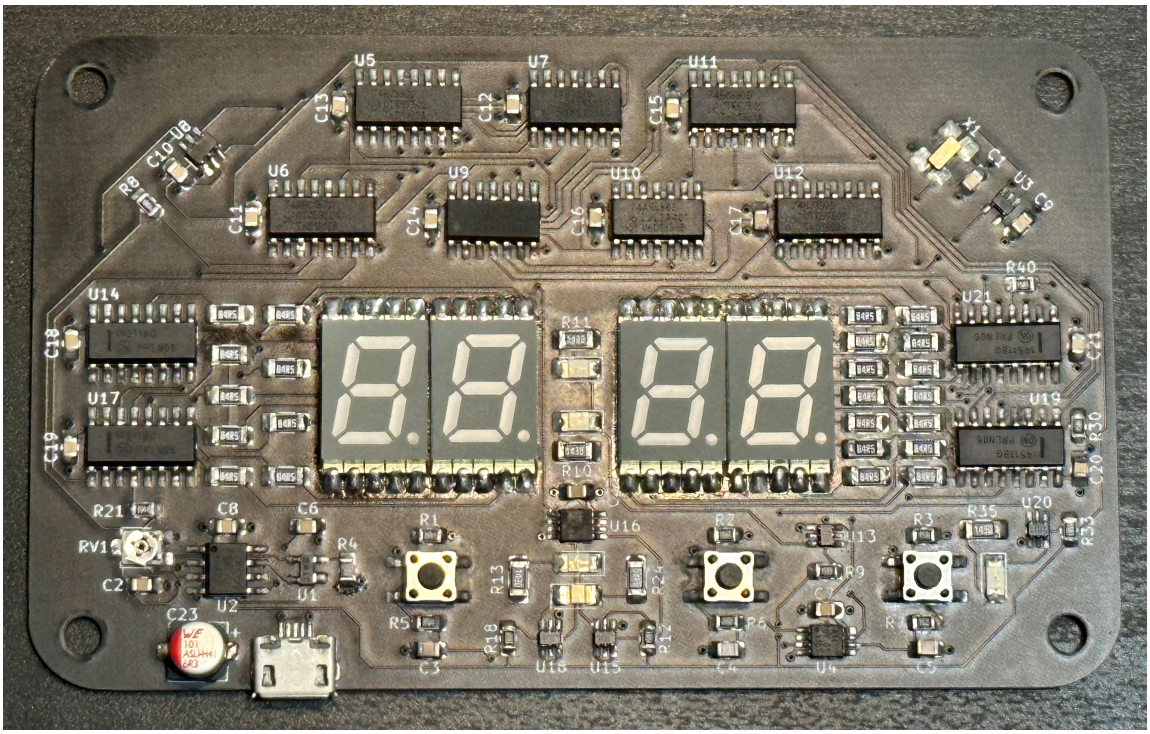


Figure 15: Finished Board

I soldered everything by hand with a TS100.