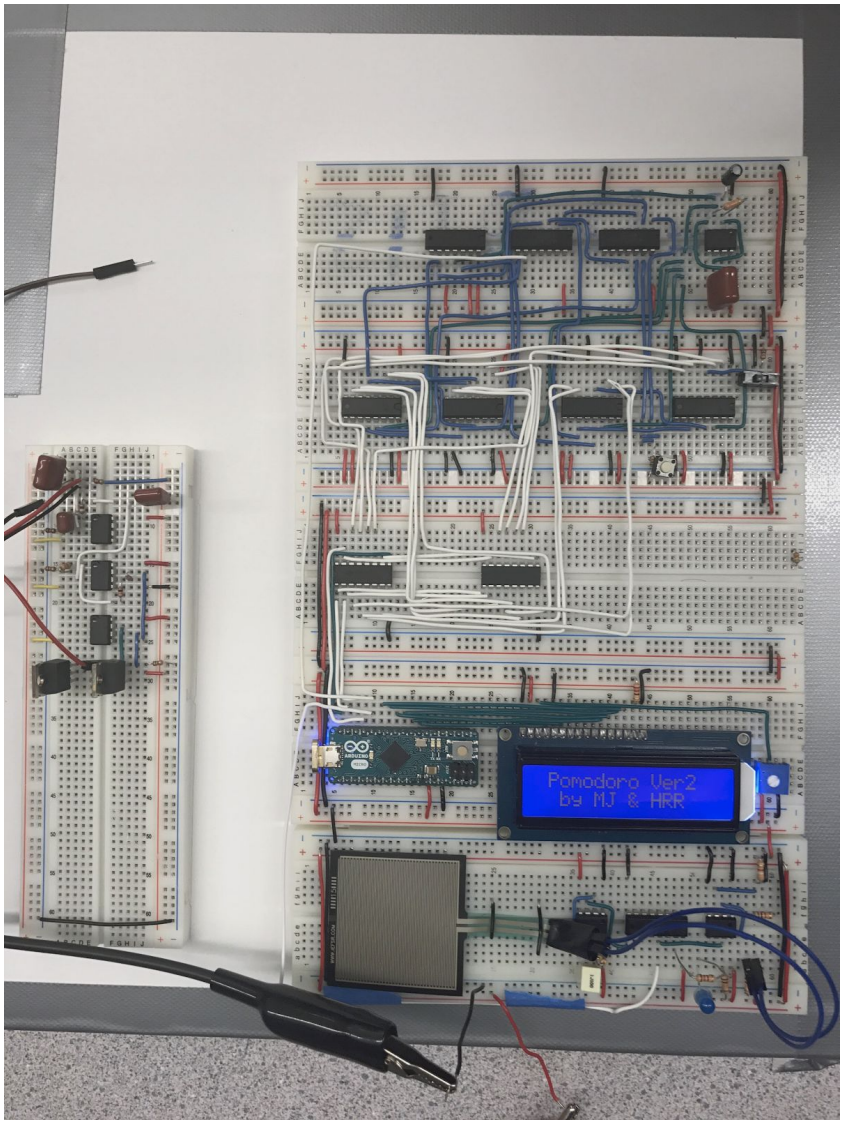


Advanced Pomodoro Timer

Mohib Jafri and Hiromu Ryan Rose

A refresh of an ancient productivity principle that breaks you free from your most distracting device-- your phone. Using force sensitive resistors to detect a phone being placed down and away from the user, a 25 minute timer counts down to zero. If you choose to pick up your phone before 25 minutes have elapsed, your countdown counts up, and you owe yourself even more time. After ceding your phone to our system during your work time, you're allowed five minutes of break before your next Advanced Pomodoro Timer cycle begins.

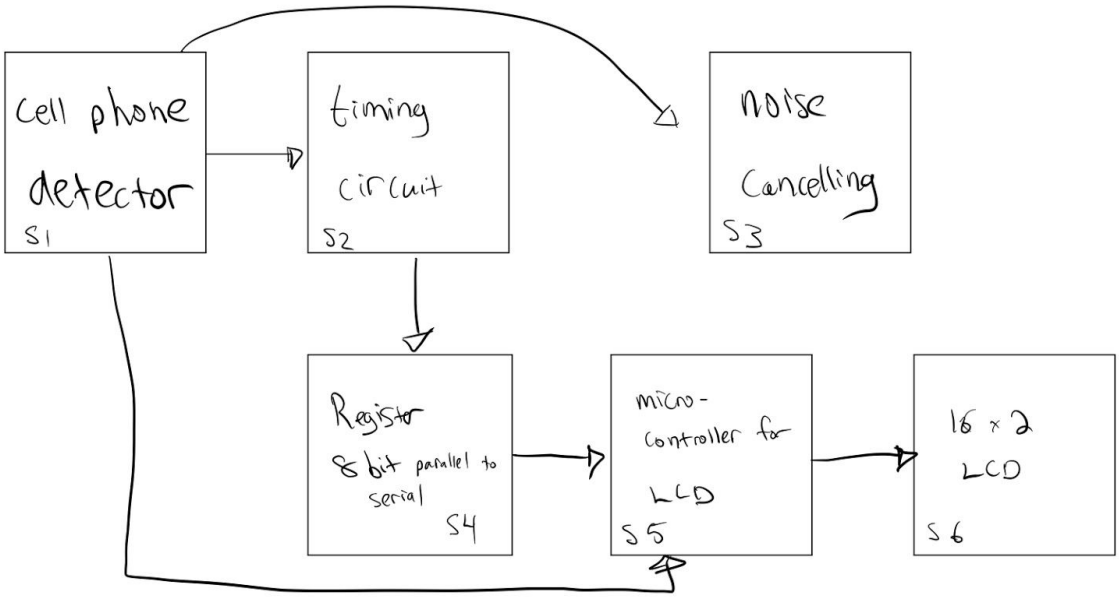
Introduction

Phones have always been a distraction when trying to finish work productively. We have both downloaded apps on our phones that reportedly increase productivity but these apps are on our phones so we are still prone to distractions from notifications. This is why we set to make a productivity timer that physically encourages us not to use our phone at all, not just within an app in the phone.

We first wanted to use a cellphone jammer to block signals coming into the phone. Most distractions arise from notifications from Internet-connected apps and calls/text messages so we wanted to jam these signals. However, we found out that the FCC (Federal Communications Commission), did not allow these jammers to be used since they could interfere with vital cell phone signals that could extend over the intended radius of disconnection. This is why we decided to have a force-sensitive resistor as a cell phone detector instead. Then, we connected this to a “pomodoro timer”-like circuit. The reason why we chose the pomodoro method for productivity was because we have constantly used this productivity method and found that it works perfectly for us. We connected this timer to an 8-bit to parallel register so that we could connect our timing circuit to the microcontroller and this to the LCD. We also wished to completely block off external distractions from people around you by having noise-cancelling headphones. We were unable to connect our noise cancelling circuit to our headphones but we were able to invert a noise, just not at the right timing to cancel the noise.

The Design

Systems Breakdown



S1, the cell phone detector circuit, sends a HI/LOW signal for if a cell phone has been detected (active HIGH). This signal, V_{CPD} (standing for cell phone detected), is used directly by both the timing circuit S2 and S5, the micro-controller driving the LCD, S6. S2 uses V_{CPD} by detecting whether the signal is HI or LOW and then counting up to the final value or away from the final value S5, implemented as a Finite State Machine, uses the V_{CPD} from S1 as the primary input for users to begin the cycle/change between different states in the work cycle. S3, the active noise cancelling subsystem, uses V_{CPD} in the positive and negative inputs of the op-amps so that if the phone is detected the noise cancelling will start since V_{CPD} would be HI and if not the op-amps wouldn't work not activating the noise cancelling. S2, the timing circuit, has 16 bits of output, 8 for minutes and 8 for seconds, that convert from parallel inputs to a serial output to be detected by S5, the micro-controller, which then displays the time increase/decrease accordingly via shifting data to S6.

Subsystems

S1: Cell Phone Detector

Design

With reference to the final design schematic below, the cell phone detector consists of U1, U2, and U3 in creating V_{CPD}, an active-high binary signal signifying if a phone is detected to be placed on the photosensitive resistor.

Testing + Build

Beginning with U1, the first step was to characterize R_{FS}, the force-sensitive resistor. It seemed that, when connected to an Ohm-meter, with nothing placed on it, the resistance measured was negligible. We found a resistance between 4k and 7k with a phone being placed, 4k for the heavier phones and 7k for the lighter phones. Through experimentation, I determined a somewhat inverse linear relationship between weight and resistance after some initial weight has been applied (Figure 1).

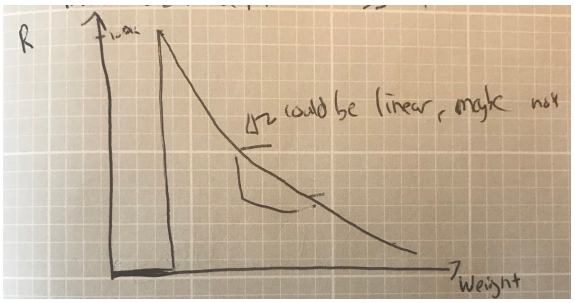


Figure 1: Postulated resistance vs. weight graph of R_{FS}. Note the maximum seemed to be at 100k.

Our measured results can be confirmed with the data sheet's specs (Figure 2), confirming a successful testing of the resistor.

The next step in U1 was to create the comparator voltage values. Using R_{FS} and a 10k resistor as a voltage divider given 5 V, I measured the voltages generated, labeled as V_+ because of its feeding to the positive terminal of the comparator (Table 1).

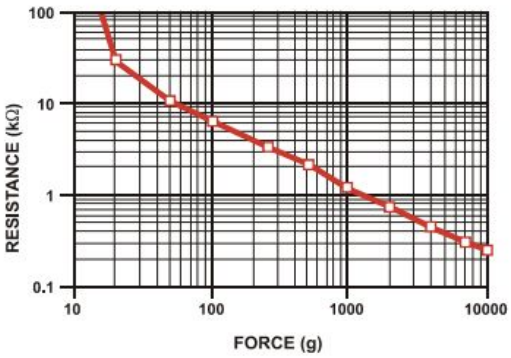


Figure 2: Resistance vs. Force

Event	R_{FS} (Ohm)	V_+ (volts)
No phone placed	0	5
Heavy phone placed	4k	1.42857
Light phone placed	7k	2.0588 V

Table 1: R_{FS} and voltages from 10k-R_{RFS} voltage divider, 10k going to ground.

The goal from here, then, was to return a high signal if any voltage lower than about 3.5 V were detected (to ensure compatibility with slightly lighter phones than the “light phone” iPhone 6s plus. This meant that V_- would need to be as close to 3.5 V as possible, which was also achieved via a voltage divider. The 1k - 3.9k chosen setup of R_2 and R_3 gives an approximate 3.75 V to compare, which is fine, as it will never trigger at *no phone placed*, and we have increased compatibility for the lighter phones of the future. The comparator U1 works as expected, giving 5V where measured when the phone was placed. However, one can see R_{FS} as behaving like a mechanical button almost with the phone bouncing on the resistor before settling. Therefore, there was a huge bouncing issue that needed to be resolved.

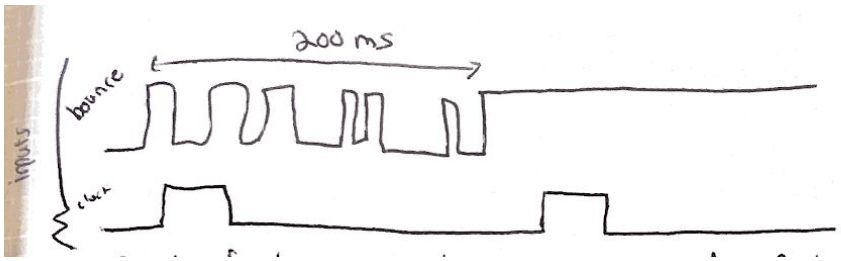


Figure 3: The bounce lasts for an average 200 ms via oscilloscope observation. A potential clock beneath that with a slightly greater period is included.

As Figure 3 suggests, the goal was to debounce using a clock-- we chose to use the flip-flop of a 74HC175. At the period of 200 ms between phone placement and a steady signal needed to not be read, we chose to create a flip flop using a clock with a 400 ms period. The slight delay expected with a longer period (at worst, less than half a second) is not as detrimental to the operation as multiple successions of a bouncing output will be. Therefore, we created a 2.5 Hz clock using the LMC555, U2, using the documentation provided for the LMC555 on the Canvas website to get a specific clock. Using the proper values for C_1, C_2, R_5, and R_6 yielded a proper clock of 2.5 Hz, which was fed into U3 as a clock with the bouncing signal as the input. Using the provided equation $f = (1.44)/(C_1/(R_6 + 2R_5))$ we chose to ensure we use less that 1 uf to avoid any polarity issue. We got a 370 ms period, which should work fine for our purposes given a 200 ms bouncing period. Our flip-flop only checks to change from low to high every 370 ms or so, meaning that we will miss the period in time in which our signal bounces and only change output at the rising edge of the 370 ms period. Therefore, using both analog and digital components, we managed to create the output of this subsystem as a clean binary signal, active high, V_{CPD}, denoting whether or not a phone has been placed on R_{FS}.

Design Narrative

We first had to choose whether to use a load cell or a force-sensitive resistor. Because we had much more practice getting voltage dividers correct after spending time characterizing R_{FS}, rather than putting the load cell in a wheatstone bridge configuration. It also made more sense intuitively to place your phone on a uniform platform rather than at the end of a metal bar. Problem solving from both a circuit design and a user experience route brings a unique perspective into how to create a good product.

After characterizing R_{FS}, it was very difficult to select what values I wanted to consider as valid for a phone. I wanted to define a set range-- say, between 4k and 7k-- that would signify a weight of a phone being placed. However, it seemed a tad too complex to spend time rejecting phones too heavy or too light, an all-to-arbitrary factor I would've had to make up myself. It was helpful reaching the decision of providing the user more flexibility to use more devices, phones being much lighter than the original "range" idea I had originally desired to pursue.

The debouncing process was relatively easy, but creating the timer was tougher. That took a lot of nearly random experimentation. I learned that it really helps to define or create constraints to ground a problem in some reality. Choosing to use non-polar capacitors framed my RC choices a lot better. I also had some difficulty figuring out if duty cycle would need to be controlled. Thankfully, I quickly realized that duty cycle is completely irrelevant and not even worth measuring (so long as it was not a nominal percentage) when the flip-flop only changes outputs as a rising-edge triggered device.

S2: Timing Circuit

Design

With reference to the final design schematic below, the timing circuit consists of U5, U6, U7, U8, U9, U10, and U11 to work as the main pomodoro timer.

<u>Seconds Timer:</u>

U5 and U6 are meant to be cascaded together to create an 8-bit timer for the seconds. This timer is solely meant to count up from 196 (base-10) to 255 (base-10) and then re-loads 96 (base-10) to create a modulo-60 timer for the number of seconds in a minute. This is achieved since RCO_{NOT} (Ripple Carry Out) goes low only when the timer reaches 255 (base-10), this is used to reload 196 (base-10) to start the 60-second timer again and to activate the $CTEN_{NOT}$ (Counter Enable) for the Minutes Timer for one clock cycle. Since this counter only counts up the $Down/Up_{NOT}$ pin is connected to ground to keep it low, and thus it will count up.

D-Flip Flop for Synchronous Load:
Since the 74HC191 that is being used for this timer has an asynchronous load, a D-Flip Flop is used so that one clock cycle passes (thus making the timer have a synchronous load) and then LD_{NOT} is asserted. This accounts for U9 in the circuit as the RCO_{NOT} of the Most Significant Bit (MSB) of the seconds timer is connected to the D input of the Flip Flop and the Q output is connected to LD_{NOT} and the $CTEN_{NOT}$ of the Least Significant Bit (LSB) of the minutes timer.

Minutes Timer:
U7 and U8 are meant to be cascaded together to create an 8-bit timer for the minutes. This timer is solely meant to count up from 226 (base-10) to 255 (base-10) and then re-loads 226 (base-10) to create a modulo-30 timer for the minutes in the pomodoro timer (25 minutes for work and 5 minutes for break). The $CTEN_{NOT}$ of the LSB is connected to the RCO_{NOT} of the MSB of the seconds timer and thus gets activated for one clock cycle. The pomodoro cycle gets reset once RCO_{NOT} goes low by reaching 255 (base-10) and reloads the value of 226 (base-10). Since this counter is bi-directional and changes whether it's counting up to the end of the pomodoro cycle or adding time to the pomodoro cycle by counting down if the phone is being used, its $Down/Up_{NOT}$ pin is connected to the inverted output of the cellphone detector. This will change the direction of the minutes timer based on whether the phone is being used and taken off the detector (0V inverted to 5V so that timer counts down) and viceversa (5V inverted to 0V so that the timer counts up).

Logic Gate for Push Button and RCO_{NOT} Loading:
U10 and U11 are a NAND (74HC00) and inverter (74HC04) package respectively. These are used so that both the Push Button to manually load the counters and so that RCO_{NOT} can load up the values of the seconds and minute timers so that there is a continuous cycle.

Build + Testing

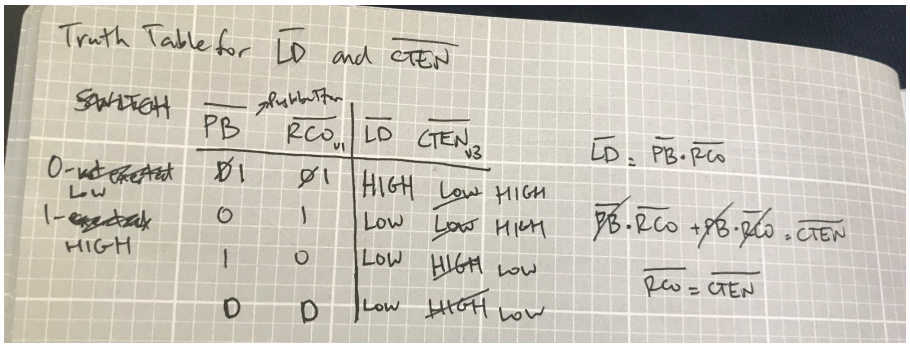
For the building a 1 Hz Clock was first made by using an LMC555 timer and a 330k resistor with a 2.2 microF capacitor to create a constant 1 Hz signal. After testing it was found that this clock generated a 1.02 Hz signal but was still within 5% of 1 Hz.

Then each of the respective timers was loaded with a preset value as mentioned in the *Design* section. 74HC191 timers were used for all 4 timers to maintain unity and less error by using the

same parts consistently and since bidirectional counters were needed for the circuit because minutes timer needed to count up or down depending on whether the phone was on or off the cell phone detector. A slide switch was used to start the timer and the $CTEN_{NOT}$ of the seconds circuit.

For testing if the timer worked correctly and to see if it reloaded correctly, a smaller resistor (33k) was used for the Clock so that it would generate a 10 Hz signal and make the whole timer 10 times faster. This way it was possible to debug easily instead of waiting 30 minutes.

To make the Logic Gate for the Push Button and RCO I (Ryan) created the following Truth Table:



Design Narrative

In the beginning, we were planning on how to make this timer without the use of a bidirectional counter since we had never used those and were unfamiliar. I ended up making a design with multiple 74HC161's but this required 12 timers just for the work timer alone since 4 timers were needed for the seconds and minutes of the actual timer, then 4 timers were needed for the penalty stopwatch seconds and minutes accumulated for using your phone during work and 4 timers were needed to load this value onto a penalty timer that counted up to the penalty value and then the normal timer would resume. This was deemed to be too repetitive and complex since the design could be simplified and cleaned up with a bidirectional counter.

Also, I realized late that the bidirectional counter had asynchronous load since my seconds timer and minutes timer wouldn't cascade properly. This was a very difficult part in the timer since I performed multiple tests by even trying to use MAX/MIN (pin 12) which did not work successfully since it would go to LOW once the LSB of the seconds timer hit 0000, and it would go HIGH once the LSB of the seconds timer hit 1111, which is not what I wanted since it was supposed to be just LOW for 1111 in LSB and 1111 in MSB and HIGH for every other second. With the help of Jim MacArthur I realized that I needed to implement a flip flop to make the counter wait a clock cycle to load the values again.

S3: Active Noise Cancelling

Design

For the noise cancelling, the components U16, U17, U18, U19, U20, and U21 were used.

U16 is used based on the V_{CPD} that U3 outputs, it inverts this to make the negative input of the op amp and uses V_{CPD} for the positive input of the op-amp. This would basically make the op-amp inputs 5V and -5V when the phone is detected by the Force Sensitive Resistor (FSR), and it would be 0V and 0V thus rendering the noise-cancelling useless if the phone is not detected by the FSR.

Then, U17 is the Electret Microphone which is used to pick up noise from the surroundings. However, this only produces about 15 mV which is V_{MIC}.

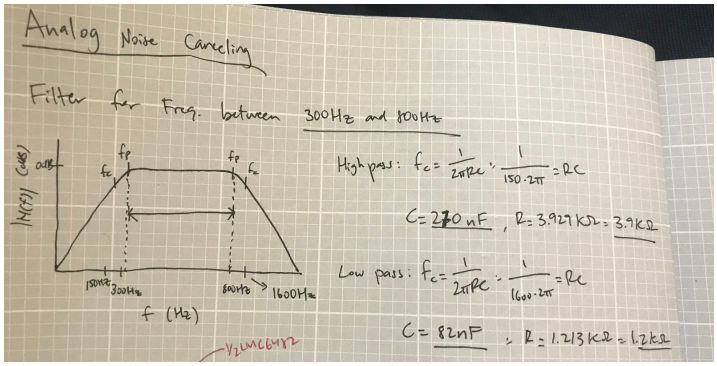
This V_{MIC} is then filtered using a high pass filter and low pass filter combination to only accept noises between 300 Hz to 800 Hz (also to test with a 440 Hz sine wave) and filter out everything else to produce V_{FILT}. The calculations for this are shown below in the Build and Testing section. This V_{FILT} is then amplified by a magnitude of 3000 to produce enough voltage, and this is V_{AMP}.

Then, this V_{AMP} is inverted to get a wave that is the opposite to the noise coming in and this will be called $V_{INVERTED}$.

After, this $V_{INVERTED}$ will be put into a totem pole MOSFET, similar to how we did in Lab 3 with the Theremin, then the negative feedback of the op-amp in U20 is connected to the sources of the MOSFETs so that hysteresis is prevented.

Build and Testing

To build this analog circuit, I (Ryan) first did calculations to make the values for the filters fit an fpass of between 300 Hz and 800 Hz. For the high pass filter the f_c is half of f_p so using 150 Hz and the formula $f_c = 1/(2\pi RC)$, the values of R = 3.9 k and C = 270 nF were determined. For the low pass filter, using the same formula but having f_c be twice f_p (1600 Hz), the values of C = 82 nF and R = 1.2 k were calculated. This is shown below in the calculations.



To set up the microphone in the beginning we used a 2.2 k resistor with a 1 microF capacitor by following the specs of the Electret (in Bibliography). Then this was passed through a unity gain buffer (U18: ½ LMC6482) and then passed through the filters mentioned previously.

For the voltage amplifier using gain we went through testing to determine how much we wanted to gain by for the voltage difference to be enough to take into account ambient noise. There was a flaw on our 0mV to 15mV noise range since that was the maximum noise area we saw when our mouth was next to the microphone but we neglected to control solely for ambient noise. This is why instead of a gain of 1000, we went for a gain of 3000 by using a 3M resistor in between V_{Amp} and V_- and 1k resistor between V_- and ground (as shown in Final Schematic).

After that we inverted the signal we got to try to cancel the noise waves by using an inverting op-amp (U19). Then this was passed through a totem pole MOSFET similar to the Push-Pull amplifier used in Lab 3 since it needs to account for the limited output current of the op amps.

We attached this to speakers instead of the actual headphones since we could not get a mechanism to connect the headphones to the circuit so we decide to connect the whole circuit to a speaker to try to noise cancel a sample 440 Hz sine wave and apply our learnings for the actual noise cancelling headphones. The 1.1 K resistor (R_{21}) is used as a stand-in so that we can test the circuit before we connect it to the speakers.

Design Narrative

We first tried to make noise-cancelling headphones for this project since we wanted to have a completely immersive experience with no distractions. However, quickly we realized that this would be very hard since it is difficult to account for all the ambient noises around with just an analog circuit and thus we realized that people in the noise-cancelling headphone industry usually use the help of microcontrollers for noise cancelling. Also, the wiring of the headphones to the circuit would get pretty complicated since it would involve connecting our headphones to the circuit and the music source at the same time.

Thus, we decided to improvise and create a noise cancelling circuit that would account for a 440 Hz sine wave and use speakers to try to cancel this noise. After talking to Mike Litchfield we

realized that we had to amplify the signal we got, filter it, invert it and then pass it through a Totem Pole MOSFET formation to account for the limited current in the output of the resistor. The easy part for me was creating the filter, the amplification parts, and the MOSFET totem pole since we had done something very similar for Theremin Lab 3. However, we realized that the noise was not getting cancelled effectively since the sine waves were not mirrored across the horizontal axis where V = 0V. The waves were slightly shifted and after talking to Mike Litchfield again we realized that we needed to use an All-Pass Filter to pass all the signals but slightly delayed after we had done the inversion, amplification, and initial filtering. This all-pass filter would account for the delay that it takes for the sound of the speakers to reach the human ear and we would use it to delay our noise cancelling signal so that the two sine waves would match and be completely mirrored and thus cause destructive interference, so that we would hear no noise.

Accounting for the delay was something that was very difficult and if we were to improve upon the project we would definitely account for this to truly cancel noise and then incorporate it into the headphones.

S4-6: 8 Bit Parallel to Serial and LCD interfacing with Microcontroller

Design

Given two sets of 8 bits (one set for minutes, one set for seconds), it would be wasteful to create 16 inputs into the Arduino to tell the time when a digital component can convert these parallel inputs into a serial output that only requires four pins of the Arduino microcontroller. U_{12}, the seconds, and U_{13}, the minutes registers are cascaded together to form one register. Using essentially a clock, denoted as POLL_DELAY_MSEC in the Arduino code, we shift in the values retrieved at the last clock cycle and use Arduino logic I made to interpret that as time. See full comments in Appendix for documentation of code.

Build/Testing

There were several iterations of the FSM, some only partly complete upon realizing the need for greater states. The final FSM, visualized:

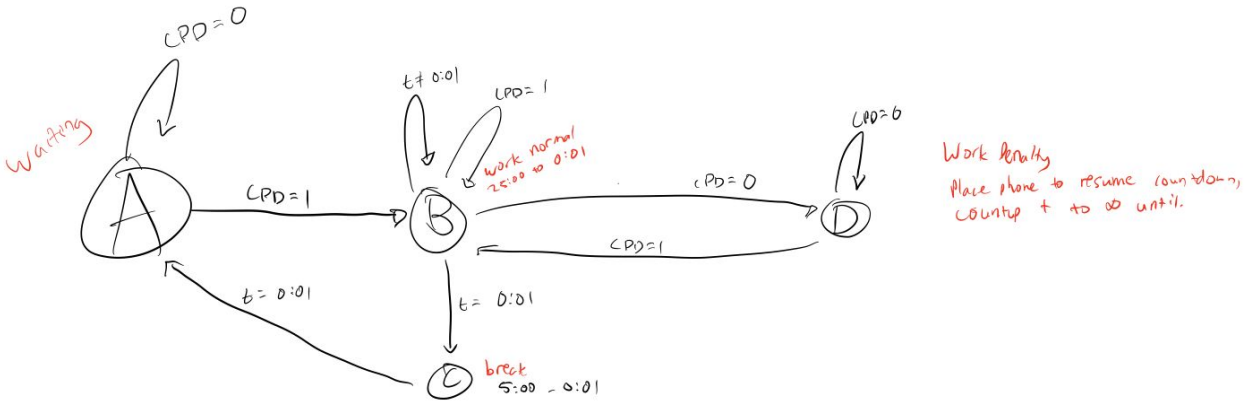


Figure 4. FSM of Arduino code to dictate user experience via 16x2 LCD.

Note that inputs that did not matter (CPD, t) at each state were omitted to see only the functional components that were being checked in the FSM code.

Design Narrative

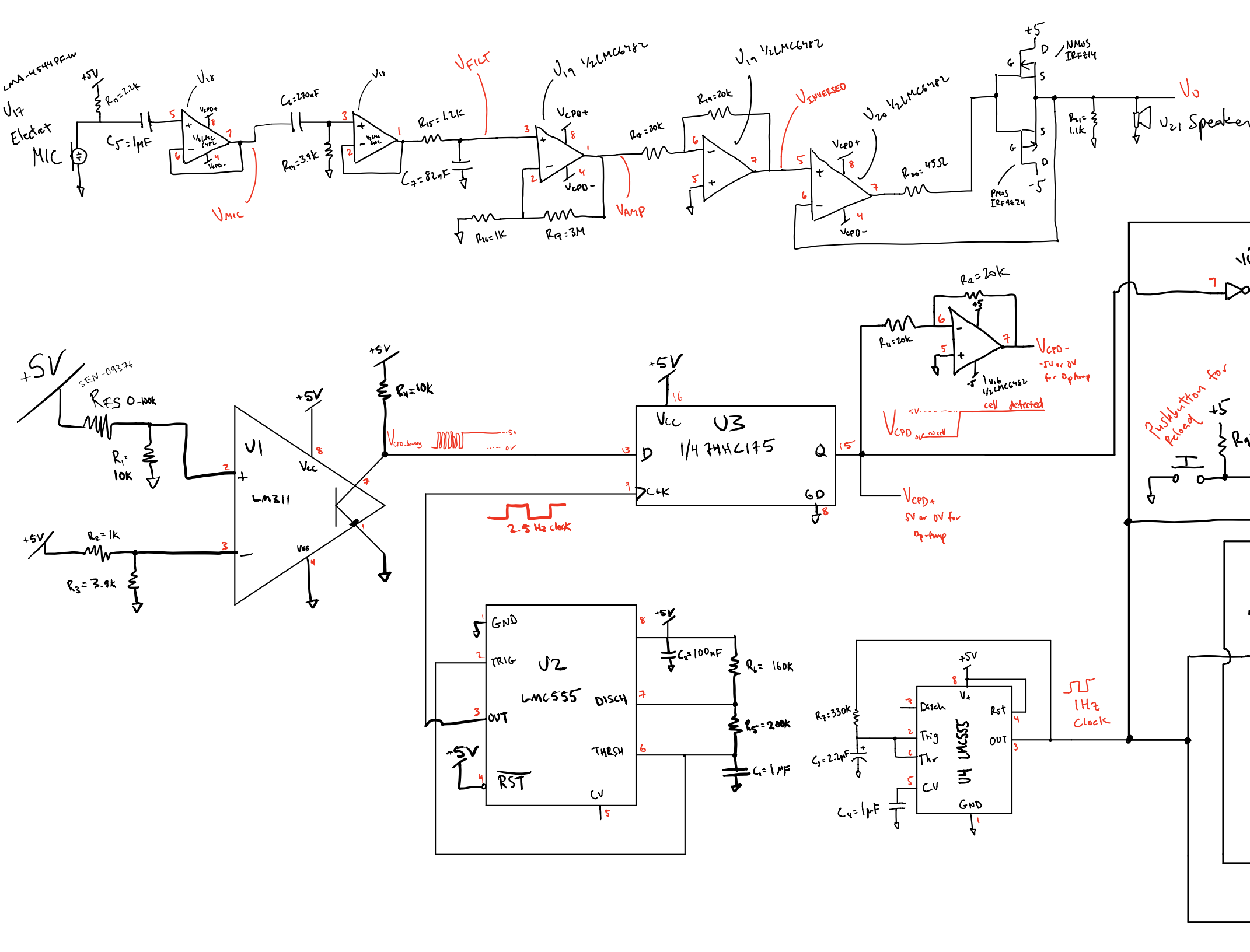
This was by far my (Mohib's) funnest part of the project. Being able to implement the circuit-design way of thinking into creating a strong FSM was very rewarding. It was most challenging to figure out how to express countdown vs. countup without repeating too much code. I'm happy I managed to create the formattedTime() helper function and recognizing how the code can be most efficiently written without rewriting seconds logic too much. It was very difficult to trouble shoot with a 25 minute clock to have to wait through. Luckily, we realized that changing our RC values can make the "25 minutes" last only a few seconds, which helped us debug our code much quicker.

Results/Design Narrative

Each subsystem held its own distinct feats of design. Please see their corresponding "Design Narrative" sections as well as the results discussed in the build/testing process.

Final Schematic

Please see the attached paper for the final schematic.


CMA-4544PF-W
+5V
R8=2.2k
C5=1μF
MIC
VMIC
C6=270nF
R14=3.9k
R15=1.2k
C7=82nF
VFILT
R16=1K
R17=3M
VAMP
R18=20k
R19=20k
VINVERSED
R20=45Ω
R21=1.1k
Speaker
VO
NMOS IRF314
PMOS IRF9Z24
+5V
SEN-09376
RFS 0-100k
R1=10K
R2=1k
R3=3.9k
LM311
R4=10K
VCPD_binary
5V
0V
R11=20k
VCPD-
-5V or 0V for Op-Amp
cell detected
VCPD
5V or no cell
VCPD+
5V or 0V for Op-amp
74HC175
2.5 Hz clock
LMC555
R6=160K
R5=200K
C1=1μF
C2=100nF
R7=330K
C3=2.2μF
C4=1μF
1Hz Clock
Pushbutton for Reload

Slide Switch SPST
Start Timer
R8=15K
+5
Makes Load pulse w/pushbutton and RCO
U10 1/14 74HC00
U11 1/6 74HC04
U5 74HC191
U6 74HC191
U7 74HC191
U8 74HC191
U9 ½ 74HC74
LD
D/Ū
CTEN
CLK
D
C
B
A
GND
Vcc
MAX/MIN
RCO
Q0
Q1
Q2
Q3
Reloads Seconds Timer
Resets Forward Cycle
Makes Load Synchronous
sec_2^7
sec_2^6
sec_2^5
sec_2^4
sec_2^3
sec_2^2
sec_2^1
sec_2^0
min_2^7
min_2^6
min_2^5
min_2^4
min_2^3
min_2^2
min_2^1
min_2^0

74HC165
U12
seconds
74HC165
U13
min
Microcontroller
U14
1602A-4 Version 2.1
16x2 LCD
U15
+5V
GND
VCC
CLK
CLK_INHIBIT
SHLD
SER
A
B
C
D
E
F
G
H
D0
D1
D2
D3
D4
D5
D6
D7
RS
R/W
E
DB4
DB5
DB6
DB7
VDD
VO

BL2
BL1
RW
Vo
16
15
5
3
$R_o = 220\Omega$
+5V
+5V
$R_{VR} = 10K$

Conclusion

This project, like this course, is best summed up as a very humbling learning experience. Doing problem sets with set answers and labs with guided help to answers pales in comparison to being faced with a blank sheet of paper and having to make something from scratch. Our decisions to drive our entire project off analog timers and sensors, interpret those analog devices in a digital means, and use micro-controllers to interact with the user was a very satisfying model to follow. It offered us the intimacy with electronic components we desired in this course, which challenged our mastery of core design skills. While our work brought a functional timer controlled by a phone's placement to increase productivity, there are several areas that we would like to improve on in the future if time allows:

First, we would address the phase shift delay issue in the noise cancelling headphones. We were able to achieve creating the exact opposite input of the sounds coming in through the electret microphone, which, if summed with the sound coming in, would achieve noise cancellation. However, due to the minute delays in our electronics (both component-wise and length of wire), we were slightly out of phase, such that we could not achieve a perfect zero-sum. After consulting with Mike Litchfield, it would have been a great option to consider creating an all-pass filter that attenuates all frequencies equally, and thus shifts signals by a specified amount. This would help us delay our noise cancelling signal enough such that we cancel out the next peak of a repetitive incoming noise.

Next, we would certainly beef up the cell-phone detection system. There are many caveats to even placing a phone down-- one can have the phone facing up, for example, leaving the user to tap away as the timer knows no better. Instead, it would be interesting to potentially use some type of light/laser sensor to detect if the surface in contact with the resistor is shiny, i.e. that the phone is faced down rather than face-up. This would help in adding another layer of security that the phone is not in use.

Last, we would expect to improve our timer in several aspects. First, to resolve existing issues, we would like to move some microcontroller logic of the breaktime to digital logic using logic gates. The issue we would want to specifically resolve is the bug between 249_{10} and 255_{10} (the break time of five minutes) where any change in V_{CPD} will alter the direction of the minute count for at *least* one clock cycle. Next, we would want to make this system truly only requiring a cell phone. Currently, we need to use a switch to start the clock (as we do not want the clock running when not in the timer sequence). Using a NAND gate that requires the V_{CPD} to be high and an output pin from the Arduino giving high if state one transitions to state two would solve this, meaning that we only begin the clock as soon as we begin the cycle itself in state two. Additionally, having to assert the load only once at the start of the cycle through a pushbutton is not ideal-- while microcontroller outputs would be the quick fix, we'd be interested in exploring a digital solution. However, after resolving these bugs, it would be great to look into modular times-- being able to make our total cycle time differ as well as the ratio between work and break. Overall, we are both very thankful for having learned a great deal about techniques of the trade in this challenging yet informative journey.

Video

Please watch our video at youtube.com/watch?v=vxJMMYhCar4, or, try the shortened link at https://goo.gl/GP5ZVt .

Bibliography

Works Cited

"Bidirectional Counter - Up Down Binary Counter." *Basic Electronics Tutorials*, Electronics-Tutorials.ws, 3 Feb. 2018, www.electronics-tutorials.ws/counter/count_4.html.

Calvert, J B. "Couners and Clocks." *University of Denver*, Du.edu, 11 July 2002, mysite.du.edu/~etuttle/electron/elect48.htm.

CUI Inc. "CMA-4544PF-W Electret Condenser Microphone." *Cui.com*, www.cui.com/product/resource/cma-4544pf-w.pdf.

Sparkfun. "FSR Guide." *Sparkfun.com*, www.sparkfun.com/datasheets/Sensors/Pressure/fsrguide.pdf+.

Texas Instruments. "74HC191 Data Sheet." *74HC191 Data Sheet*, www.ti.com/lit/ds/symlink/cd54hc190.pdf.

University of Sydney. "Synchronous Up-Down Counters." *USYD Electrical Engineering*, Usyd.edu, www.ee.usyd.edu.au/tutorials/digital_tutorial/part2/counter07.html.

Thank you to Paul Horowitz, Mike Litchfield, Jim MacArthur, David Abrams, and the ES52 CAs for all their help with the timer and noise-cancelling.

Appendix

See attached code, *Advanced Pomodoro Timer*.

```
/*
 * Advanced Pomoduro Timer
 *
 * Created by: Mohib Jafri and Ryan Rose
 * Description: FSM implemented to determine what to display to
the user via a 16x2 LCD in a Pomodoro Cycle. Takes into account
work time, work penalty events, break times, and how the time
count changes from each of those states.
 * Inspiration from LiquidCrystal Library "LCD_Beginning" and
Arduino "ShiftIn" and David Abrams "FSM Skeleton"
 */

 // Necessary libraries
 #include <LiquidCrystal.h>

 // -------- PINOUTS --------

 // LCD PINS
const int RS_PIN = 2;
const int ENABLE_PIN = 3;
const int DB4_PIN = 5;
const int DB5_PIN = 4;
const int DB6_PIN = 6;
const int DB7_PIN = 7;

// 74HC165 SHIFT REGISTER PINS
int LOAD_PIN        = 11;                                  //
Connects to Parallel load pin the 165
int CLOCK_ENABLE_PIN  = 9;                                 //
Connects to Clock Enable pin the 165
int DATA_PIN         = 12;                                 //
Connects to the Q7 pin the 165
int SR_CLOCK_PIN        = 10;                             //
Connects to the Clock pin the 165

// LCD-SPECIFIC VARIABLES
```

```
LiquidCrystal lcd(RS_PIN, ENABLE_PIN, DB4_PIN, DB5_PIN,
DB6_PIN, DB7_PIN);
byte printOffset; // For writeToLCD(): how much to offset based
on justify.
char tjustify[2]; // Left or Center, L/R...
char bjustify[2]; // For writeToLCD()
String tmessage; // For writeToLCD(): 16 byte max top message.
String bmessage; // For writeToLCD(): 16 byte max bottom
message.
String text; // Arbitrary text dump
//String displayTime; // This variable will be modified a lot
as
byte minutes;       // formattedTime() for ease of reading
minutes
byte seconds;       // formattedTime() for ease of reading
seconds

// 74HC165-SPECIFIC VARIABLES
const int NUMBER_OF_SHIFT_CHIPS = 2;                  // How
many shift register chips are daisy-chained.
const int DATA_WIDTH = NUMBER_OF_SHIFT_CHIPS * 8;     // Width
of data (how many ext lines).
const int PULSE_WIDTH_USEC = 5;                       // Width
of pulse to trigger the shift register to read and latch.
const int POLL_DELAY_MSEC = 100;                     //
Optional delay between shift register reads.
#define BYTES_VAL_T unsigned int
BYTES_VAL_T pinValues;
BYTES_VAL_T oldPinValues;

// FSM Variables
const int FSM_FREQ = 10;            // set to frequency of
state machine clock in Hz (max 500Hz)
const int FSM_TIME = 1000/FSM_FREQ; // Milliseconds per state
cycle (do not change this variable)
unsigned long CycleStart;
```

```
int CurState;                       // Holds current/next state

// Cell Phone Detection Variables
const int CPD_PIN = 8;              // Cell phone detected pin.

 void setup(){
    /// FOR DEBUGGING
    Serial.begin(9600);

    //----------- Cell Phone Detection
Setup----------------------
    pinMode(CPD_PIN, INPUT);

    //----------LCD-RELATED SETUP & INTRO (not part of
FSM)-----------
    lcd.begin(16, 2); // Set up the LCD's number of columns and
rows.
    lcd.noCursor();   // Hides ugly cursor.
    introMessage();   // Pretty welcome screen.

    // ----------SHIFT REGISTER RELATED SETUP---------------

    /* Initialize our digital pins...*/
    pinMode(LOAD_PIN, OUTPUT);
    pinMode(CLOCK_ENABLE_PIN, OUTPUT);
    pinMode(SR_CLOCK_PIN, OUTPUT);
    pinMode(DATA_PIN, INPUT);

    digitalWrite(SR_CLOCK_PIN, LOW);
    digitalWrite(LOAD_PIN, HIGH);

    /* Read in and display the pin states at startup.*/
    pinValues = readShiftRegs();
    oldPinValues = pinValues;

    //-----------------FSM RELATED SETUP----------------
```

```
    CurState = 1;         // Initialize for first state in the
program loop

 }

void old_loop(){
    // READS SHIFT REGISTER, WRITES IT TO LCD
    /* Read the state of all zones.*/
    pinValues = readShiftRegs();

    /* If there was a chage in state, display which ones
changed.*/
    if(pinValues != oldPinValues)
    {
        //Serial.print("*Pin value change detected*\r\n");
        lcd.clear();
        lcd.setCursor(0, 1);
        lcd.print(counterValues("sec"));
        oldPinValues = pinValues;
    }

    delay(POLL_DELAY_MSEC);
}

void loop()
{
    CycleStart = millis();           // get time we started
this FSM cycle
    // first do anything we need to do before every state cycle

    switch (CurState){
        case 1:       // Reset state. Waits for phone to be
placed before beginning timer.
            writeToLCD("C","Place phone","C","to begin work!");

            if (cellPhoneDetected())
```

```
                CurState = 2;

            else
                CurState = 1;
            break;

        case 2:       // WorkNormal state. Shows time countdown
25 min to 0 min until work is done.



            writeToLCD("C","Work remaining:","C",
formattedTime("wn")); // Shows time in work-normal
configuration.

            if (cellPhoneDetected())
                CurState = 2;

            // Go to WorkPenalty state if phone taken off
platform when it should be there.
            if (!cellPhoneDetected())
                CurState = 4;

            Serial.println("Case 2, time is: " +
formattedTime("wn"));
            // When work is over, go to BreakNormal state.
            if (formattedTime("wn") == "0:01")
                CurState = 3;

            break;

        case 3:     // BreakNormal State. Shows time countdown
5 min to 0 min until break is over.
            writeToLCD("C", "Well done!", "L", ("Break: " +
formattedTime("bn")));

            // Resets to start of FSM if we reach the end of
```

```
the breaktime.
            if (formattedTime("bn") == "0:01")
                CurState = 1;

            else
                CurState = 3;

            break;

        case 4:     //WorkPenalty
            writeToLCD("C", "PLACE PHONE BACK!", "C", ("Work: "
+ formattedTime("wp")));

            // Go back to work normal if phone placed back
            if (cellPhoneDetected())
                CurState = 2;

            // Stay here until user puts phone back.
            else
                CurState = 4;
            break;

        case 5: // Unused currently
            if (cellPhoneDetected())
                CurState = 2;

            else
                CurState = 4;
            break;
        case 6:
            if (cellPhoneDetected())
                CurState = 2;

            else
                CurState = 4;
            break;
```

```
        default:{
        }
    } // end of switch statement


   // Here we can add anything that needs to get done after
every FSM cycle
   pinValues = readShiftRegs();
   Serial.print("CurState is " + CurState);


   // Finally we need to wait one cycle before entering the
next state (this simulates the FSM clock)
   while (millis() < (CycleStart + FSM_TIME)) {
   }                                                    // wait one
FSM cycle
}




// ----------- CELL PHONE DETECTION HELPER FUNCTIONS
----------------------
boolean cellPhoneDetected(){
    return(digitalRead(CPD_PIN));       // Returns true if 5V,
meaning cell phone is detected on surface.
}


// ------------74HC165 SHIFT REGISTER HELPER FUNCTIONS
---------------------


/* This function is essentially a "shift-in" routine reading the
 * serial Data from the shift register chips and representing
 * the state of those pins in an unsigned integer (or long).*/
```

```
BYTES_VAL_T readShiftRegs(){
    long bitVal;
    BYTES_VAL_T bytesVal = 0;

    // Trigger a parallel Load to latch the state of the data
lines,
    digitalWrite(CLOCK_ENABLE_PIN, HIGH);
    digitalWrite(LOAD_PIN, LOW);
    delayMicroseconds(PULSE_WIDTH_USEC);
    digitalWrite(LOAD_PIN, HIGH);
    digitalWrite(CLOCK_ENABLE_PIN, LOW);

    /* Loop to read each bit value from the serial out line
     * of the SN74HC165N.
    */
    for(int i = 0; i < DATA_WIDTH; i++)
    {
        bitVal = digitalRead(DATA_PIN);

        /* Set the corresponding bit in bytesVal.
        */
        bytesVal |= (bitVal << ((DATA_WIDTH-1) - i));

        /* Pulse the Clock (rising edge shifts the next bit).
        */
        digitalWrite(SR_CLOCK_PIN, HIGH);
        delayMicroseconds(PULSE_WIDTH_USEC);
        digitalWrite(SR_CLOCK_PIN, LOW);
    }

    return(bytesVal);
}

byte counterValues(char section[4]){

    if (section == "sec"){
```

```
        return (pinValues>>8);
    }

    if (section == "min"){
        return (pinValues);
    }
}



// ------------ LCD HELPER FUNCTIONS ------------------

// Creates a short startup screen before beginning Pomodoro
Cycle reset stage.
void introMessage(){
    lcd.setCursor(1,0);
    lcd.print("Pomodoro Ver2");
    lcd.setCursor(2,1);
    lcd.print("by MJ & HRR");
    delay(2500);
    lcd.clear();
}

void writeToLCD(char t_justify[2], String tmessage, char
tjustify[2],  String bmessage){

    lcd.clear();

    //-------------- Top Message ------------------------

    if (tjustify == "L")
        printOffset = 0;

    if (tjustify == "C")
        printOffset = (16 - (tmessage.length()))/2; // Decent
center-justify for an even-numbered display.
```

```
    lcd.setCursor(printOffset,0);
    lcd.print(tmessage);

    // --------------- Bottom Message -------------------------
    if (bjustify == "L")
        printOffset = 0;

    if (bjustify == "C")
        printOffset = (16 - (bmessage.length()))/2; // Decent
center-justify for an even-numbered display.

    lcd.setCursor(printOffset,1);
    lcd.print(bmessage);
    return;

}

// Takes the raw inputs coming from shift register and returns
the right display time given either work or break normal "bn",
work normal "wn" or work penalty "wp"
String formattedTime(char sequence[3]){

    // Work Normal
    if (sequence == "wn"){
        minutes = 225 - counterValues("min") + 25;
        seconds = 195 - counterValues("sec") + 60;
    }

    // Work Penalty
    if (sequence == "wp"){
        minutes = 225 - counterValues("min") + 25;
        seconds = 60 - 255 + counterValues("sec");
    }

    // Break Normal
```

```
    if (sequence == "bn"){
        minutes = 250 - counterValues("min") + 5;
        Serial.println(counterValues("min"));
        seconds = 195 - counterValues("sec") + 60;
    }


    // Final formatting, padding of zeros on seconds.
    if (seconds < 10)
        return (String(minutes) + ":" + "0" + String(seconds));

    else
        return (String(minutes) + ":" + String(seconds));


}
```