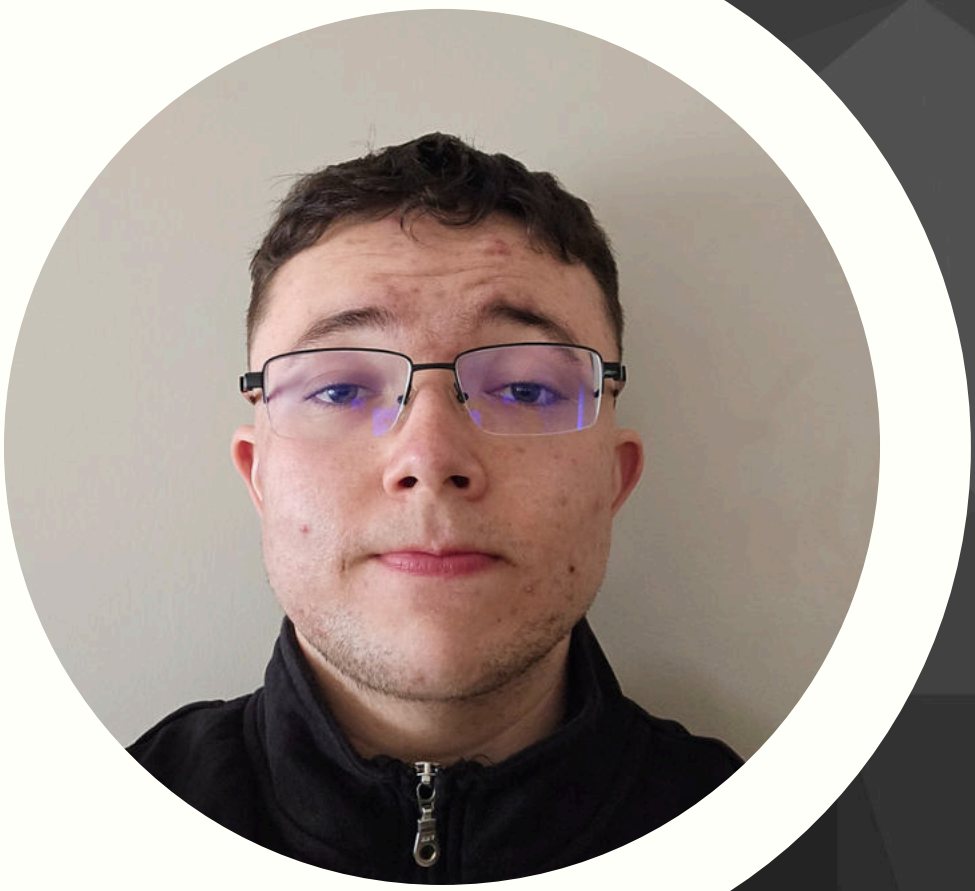
# PORT FOLIO

By Nick Rathfelder

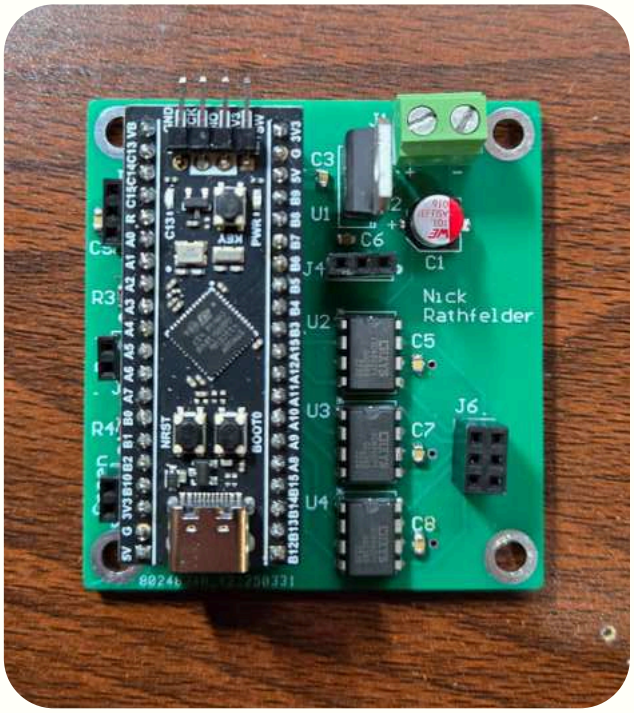


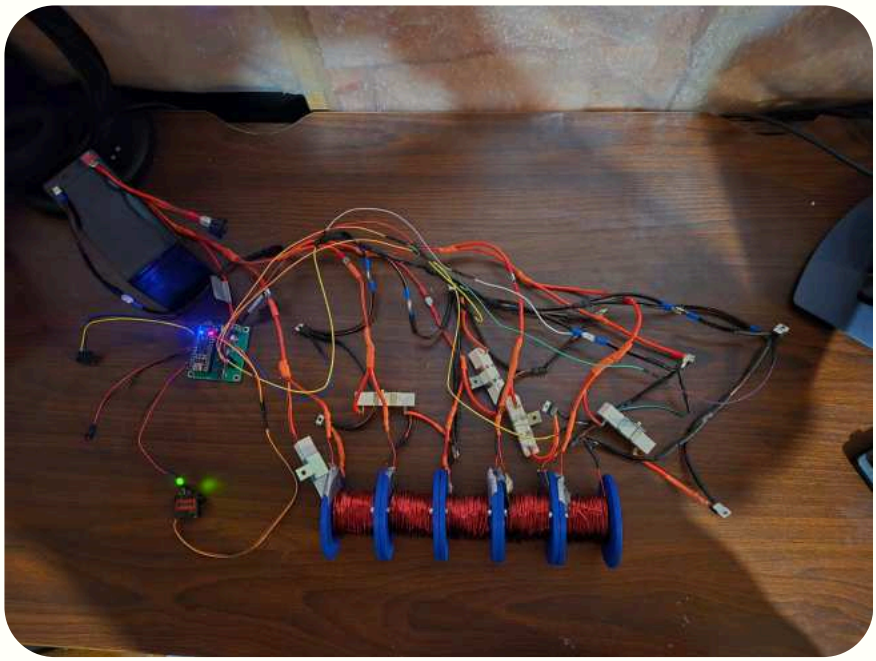
Electrical Engineering
2022 - 2027

# PROJECTS

# ELECTROMAGNETIC LAUNCHER

STM32
Interface PCB

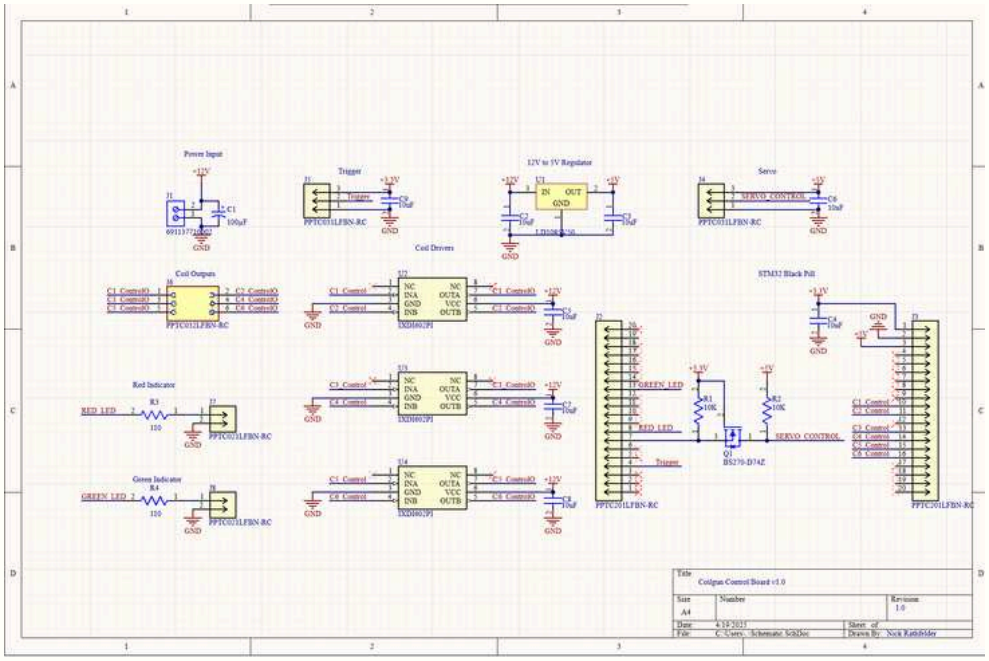


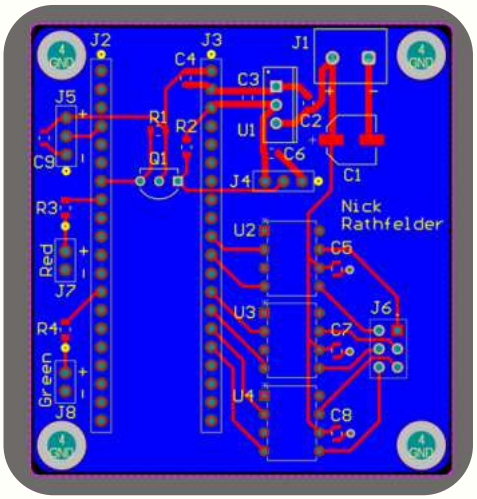



Final Prototype

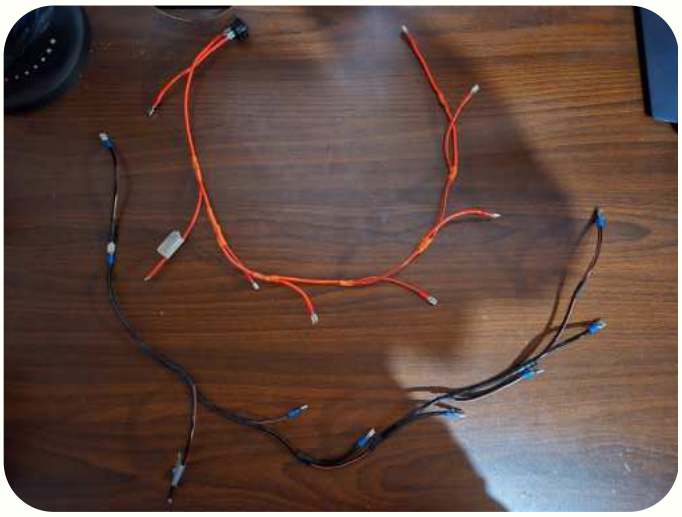
# ELECTRICAL

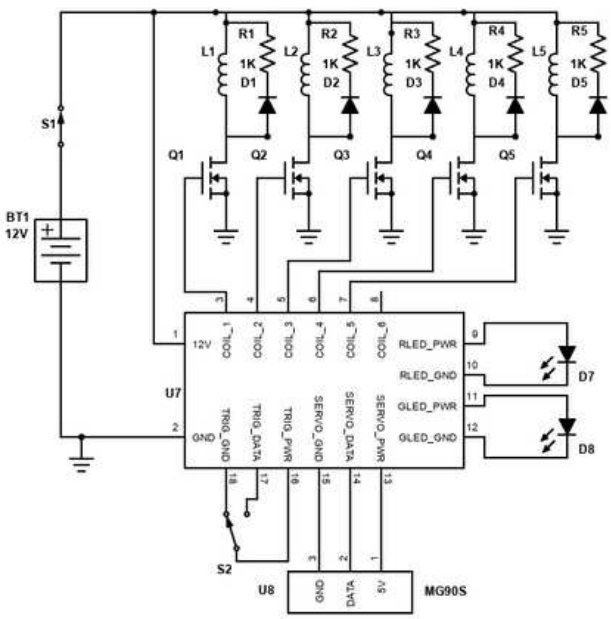
## Interface PCB

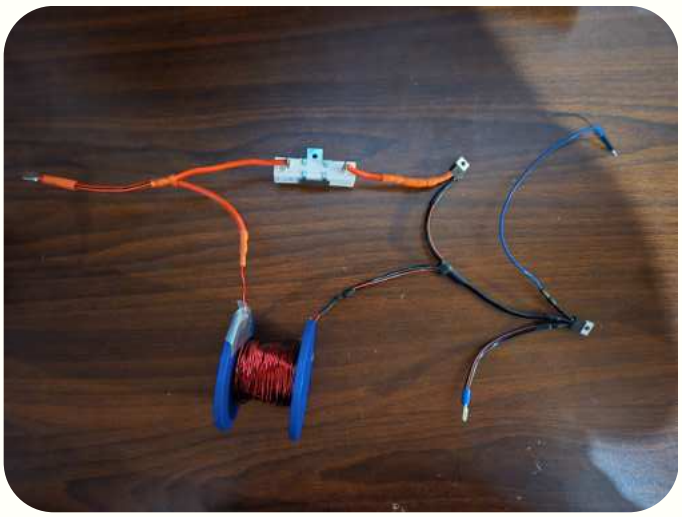






## Wire Harness

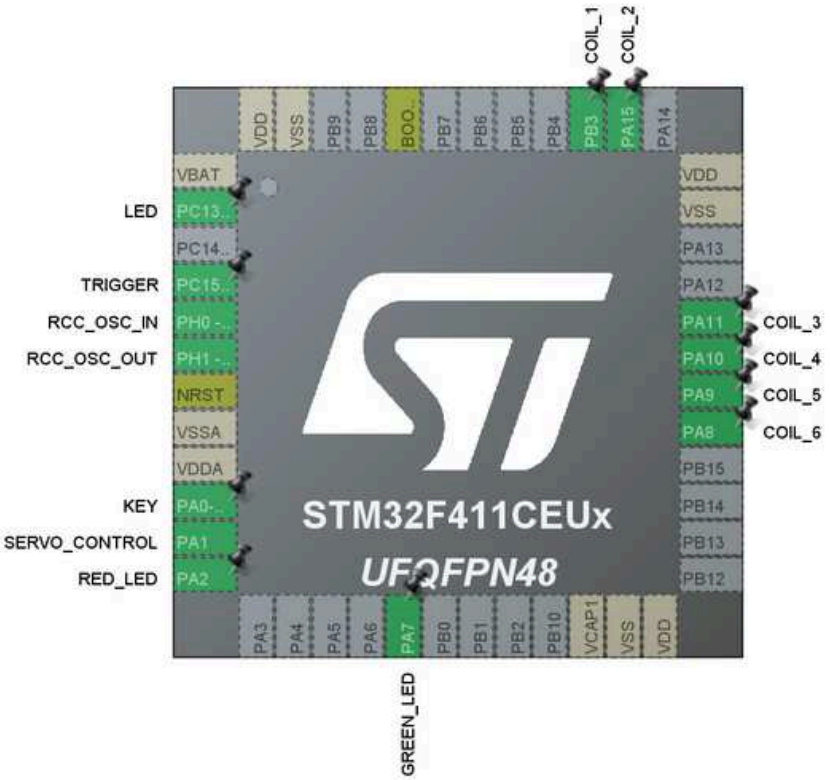




# EMBEDDED

IOC File

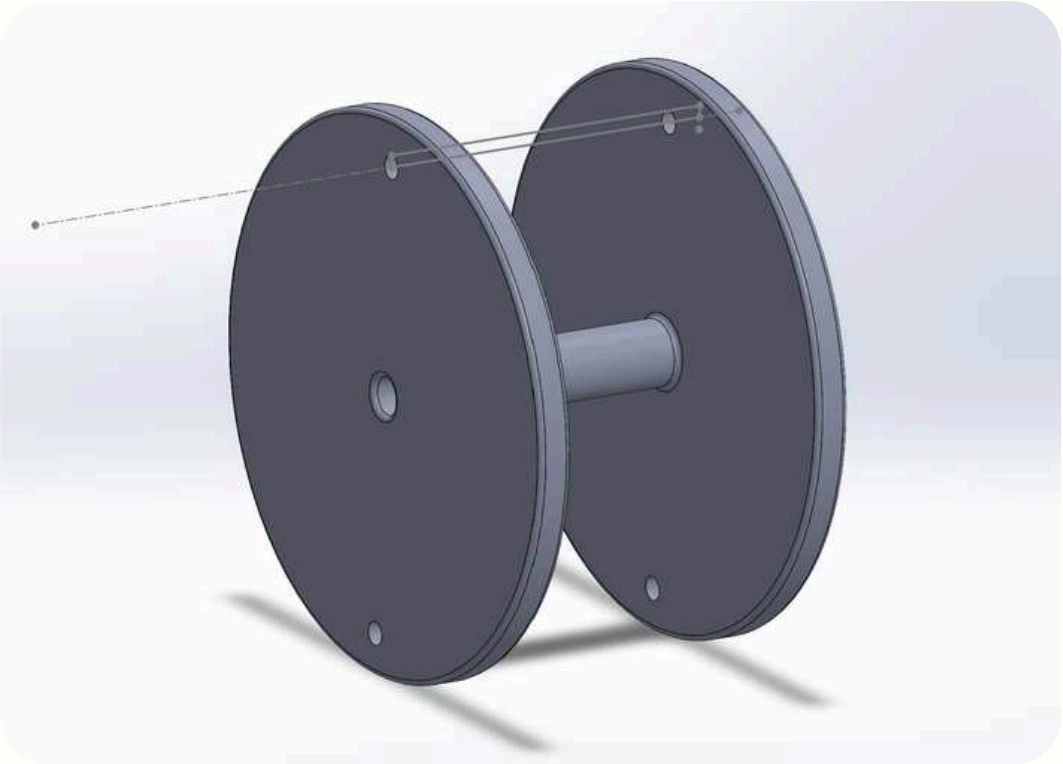


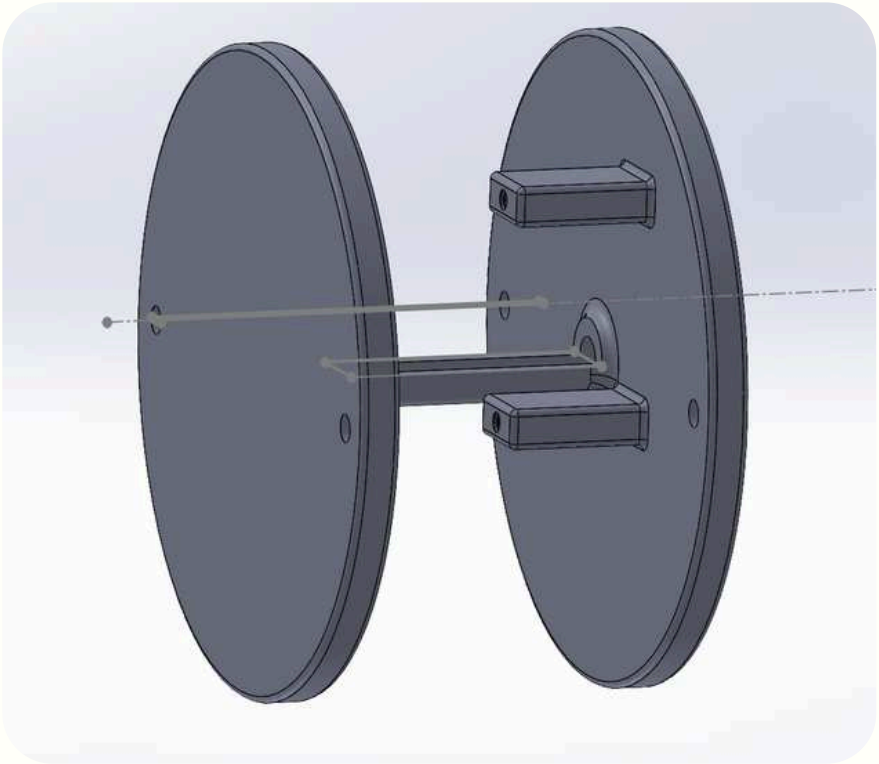

```c
while (1)
{
    trigger_state = HAL_GPIO_ReadPin(TRIGGER_GPIO_Port,TRIGGER_Pin);
    if (trigger_state == GPIO_PIN_SET)    {
        HAL_GPIO_WritePin(RED_LED_GPIO_Port, RED_LED_Pin,GPIO_PIN_SET);
        HAL_GPIO_WritePin(GREEN_LED_GPIO_Port,GREEN_LED_Pin,GPIO_PIN_RESET);
        TIM2->CCR2 = servo_open;
        HAL_Delay(500);
        HAL_GPIO_WritePin(COIL_1_GPIO_Port, COIL_1_Pin,GPIO_PIN_SET);
        HAL_Delay(10);
        HAL_GPIO_WritePin(COIL_1_GPIO_Port, COIL_1_Pin,GPIO_PIN_RESET);
        HAL_GPIO_WritePin(COIL_2_GPIO_Port, COIL_2_Pin,GPIO_PIN_SET);
        HAL_Delay(1);
        HAL_GPIO_WritePin(COIL_2_GPIO_Port, COIL_2_Pin,GPIO_PIN_RESET);
        HAL_GPIO_WritePin(COIL_3_GPIO_Port, COIL_3_Pin,GPIO_PIN_SET);
        HAL_Delay(1);
        HAL_GPIO_WritePin(COIL_3_GPIO_Port, COIL_3_Pin,GPIO_PIN_RESET);
        HAL_GPIO_WritePin(COIL_4_GPIO_Port, COIL_4_Pin,GPIO_PIN_SET);
        HAL_Delay(1);
        HAL_GPIO_WritePin(COIL_4_GPIO_Port, COIL_4_Pin,GPIO_PIN_RESET);
        HAL_GPIO_WritePin(COIL_5_GPIO_Port, COIL_5_Pin,GPIO_PIN_SET);
        HAL_Delay(1);
        TIM2->CCR2 = servo_closed;
        HAL_Delay(2000);
        HAL_GPIO_WritePin(RED_LED_GPIO_Port, RED_LED_Pin,GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GREEN_LED_GPIO_Port,GREEN_LED_Pin,GPIO_PIN_SET);
    }
    else   {
        HAL_GPIO_WritePin(COIL_1_GPIO_Port, COIL_1_Pin,GPIO_PIN_RESET);
        HAL_GPIO_WritePin(COIL_2_GPIO_Port, COIL_2_Pin,GPIO_PIN_RESET);
        HAL_GPIO_WritePin(COIL_3_GPIO_Port, COIL_3_Pin,GPIO_PIN_RESET);
        HAL_GPIO_WritePin(COIL_4_GPIO_Port, COIL_4_Pin,GPIO_PIN_RESET);
        HAL_GPIO_WritePin(COIL_5_GPIO_Port, COIL_5_Pin,GPIO_PIN_RESET);
        HAL_GPIO_WritePin(COIL_6_GPIO_Port, COIL_6_Pin,GPIO_PIN_RESET);
        HAL_GPIO_WritePin(GREEN_LED_GPIO_Port, GREEN_LED_Pin,GPIO_PIN_SET);
        HAL_GPIO_WritePin(RED_LED_GPIO_Port, RED_LED_Pin,GPIO_PIN_RESET);
        TIM2->CCR2 = servo_closed;
    }
```

main.c

# MECHANICAL

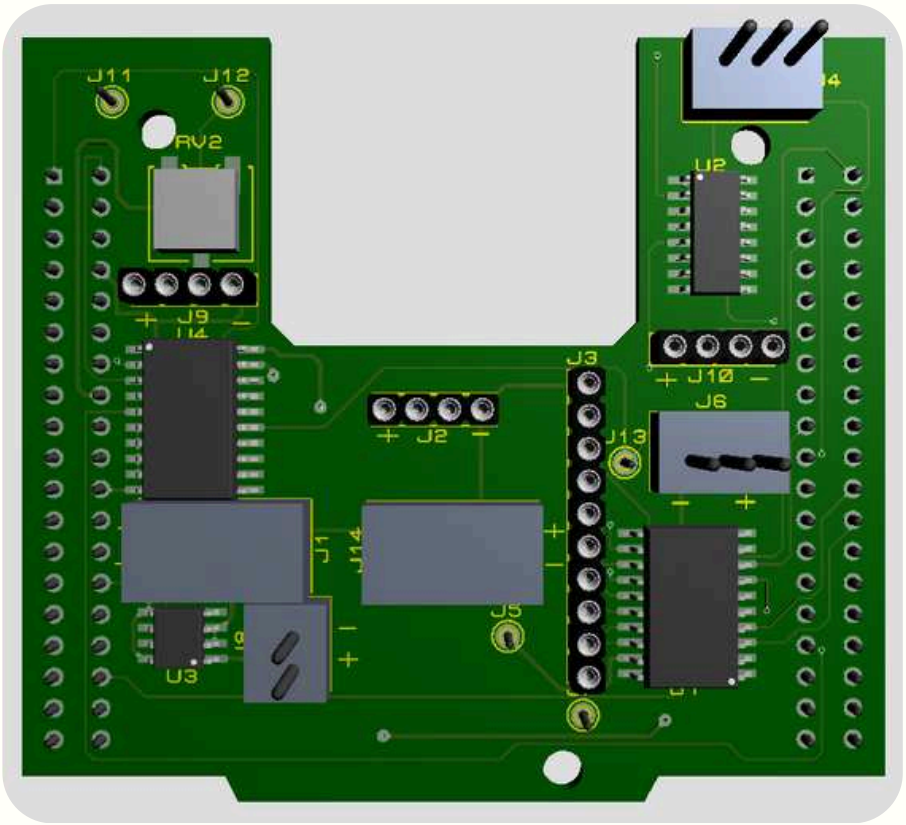
Primary Coil Design



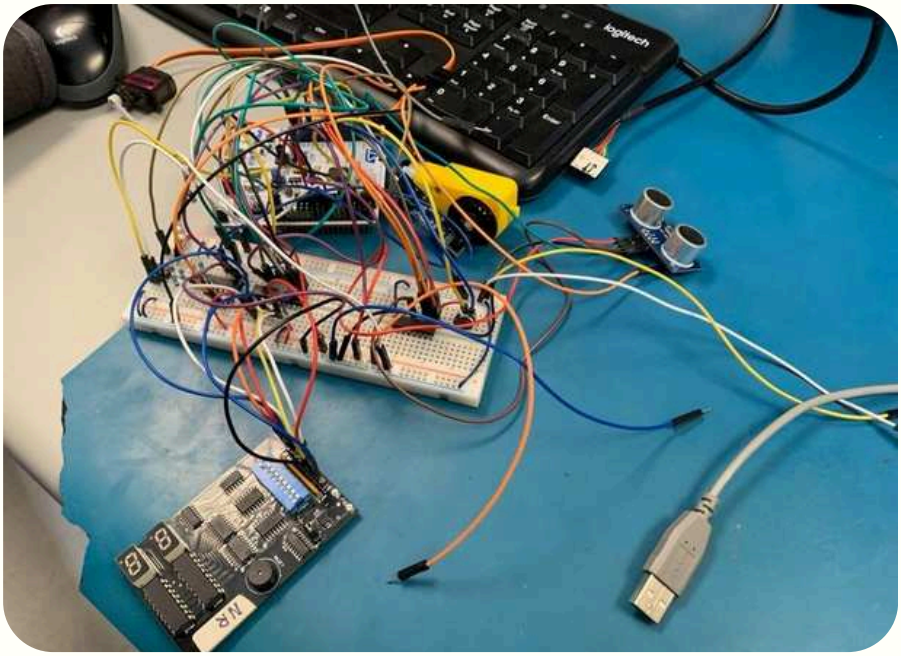


Loading Coil Design

# WATER RESERVOIR CONTROL SYSTEM

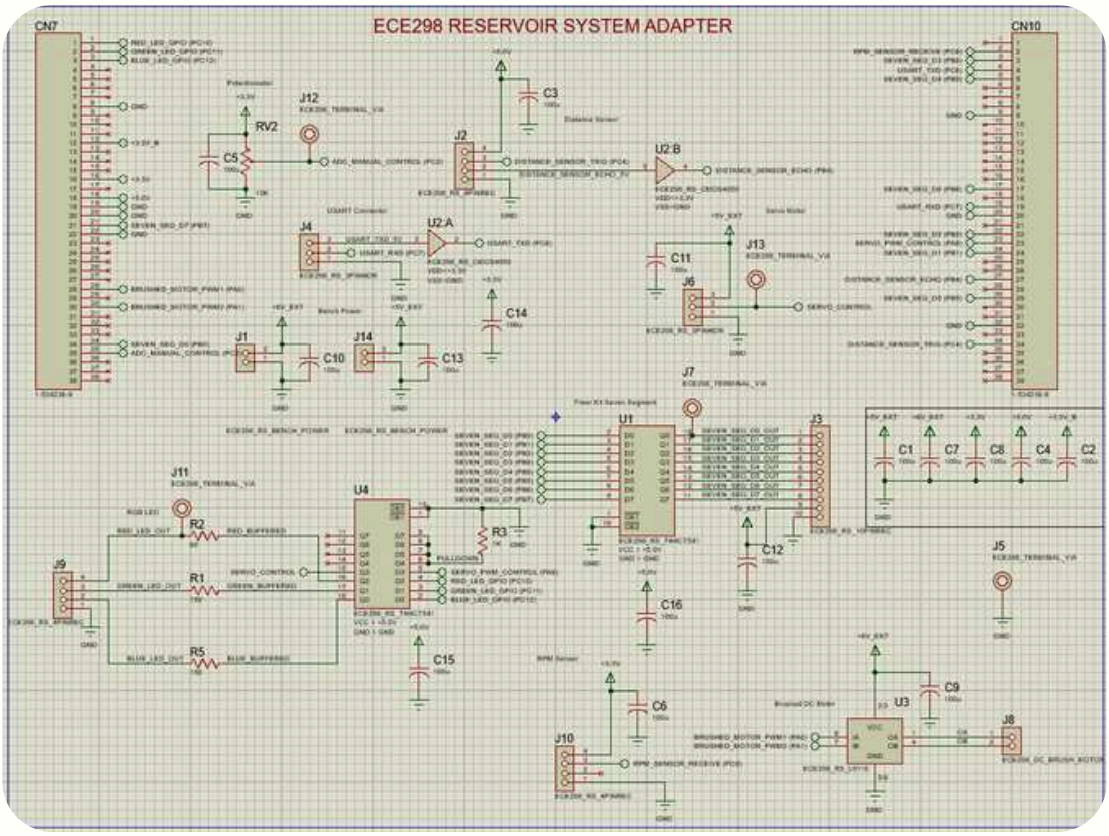



MCU Interface PCB

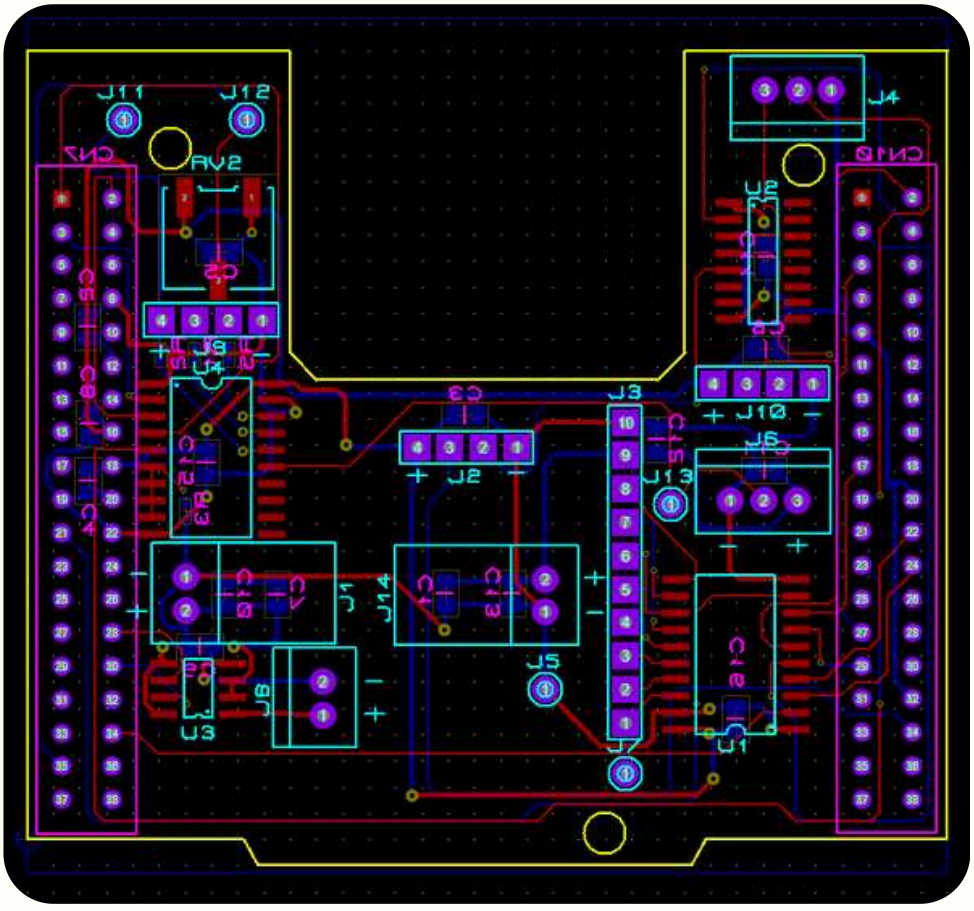



Functional Prototype

# ELECTRICAL



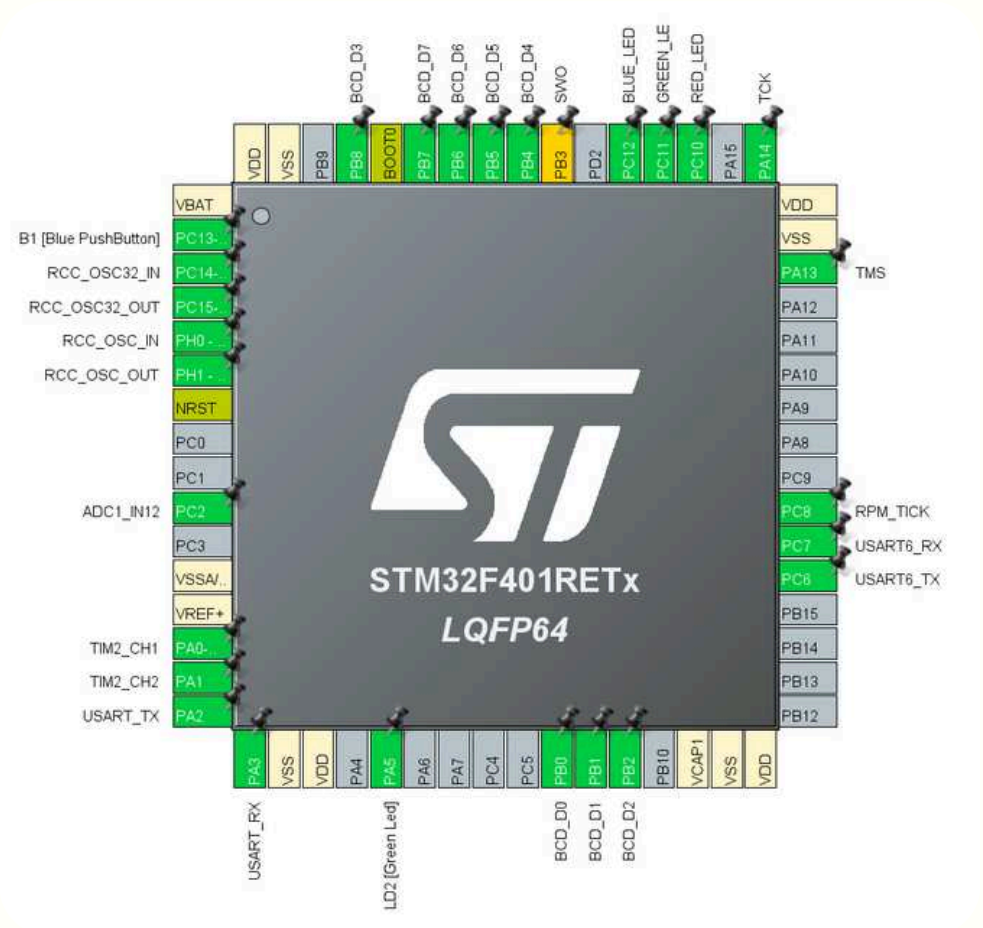

Interface PCB Schematic




Interface PCB Layout

# EMBEDDED

IOC File




main.c

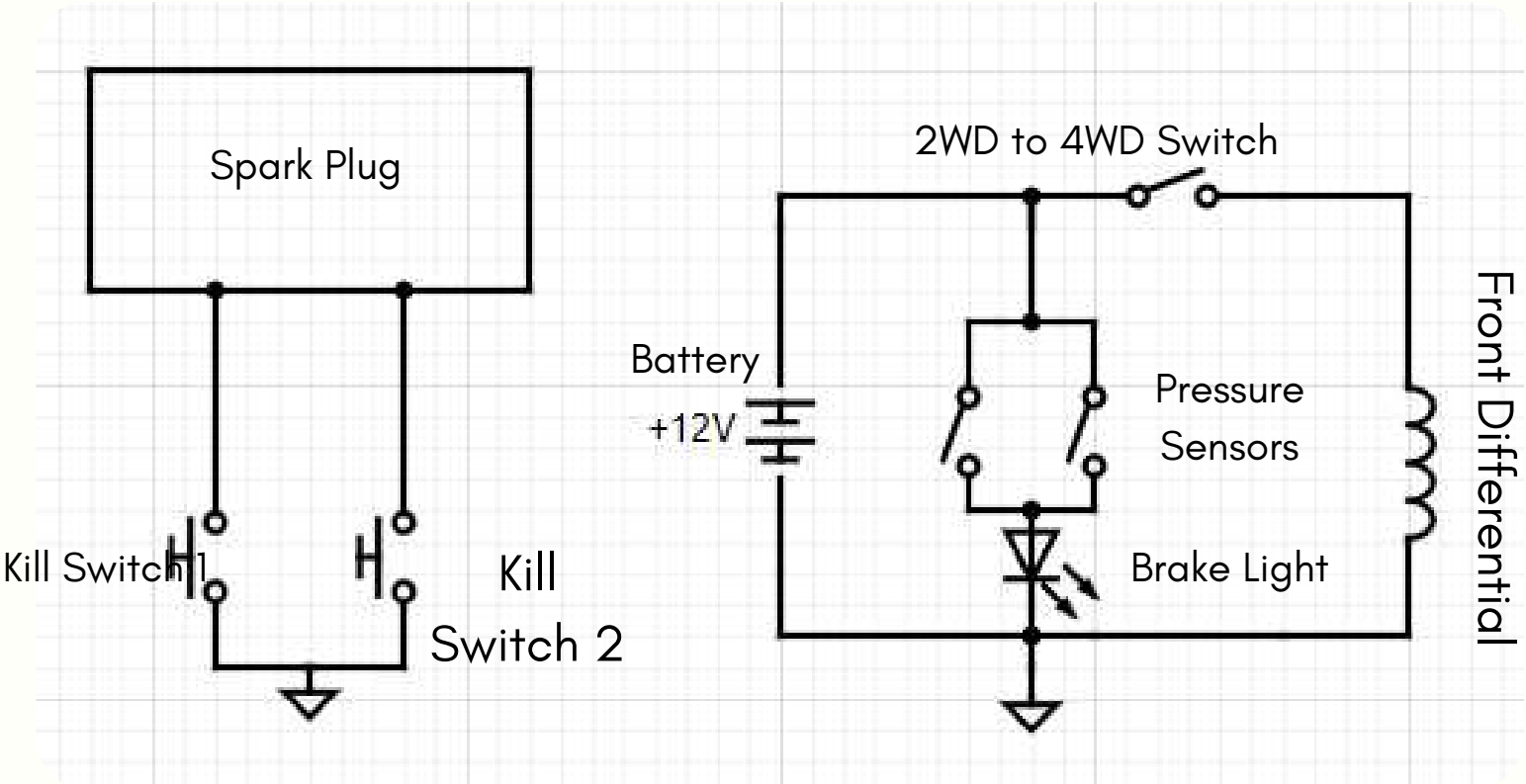
```c
while (1)
{
  //potentiometer code
  ADC_Select_CH(9);
  HAL_ADC_Start(&hadc1);
  HAL_ADC_PollForConversion(&hadc1, 1000);
  uint8_t ADC_CH9 = HAL_ADC_GetValue(&hadc1);
  HAL_ADC_Stop(&hadc1);


  // Potentiometer with dc motor code
  TIM3_DCVAL = ((float)ADC_CH9/255)*60000;
  TIM3->CCR1 = (int)TIM3_DCVAL;

  sprintf((char*)txd_msg_buffer, "RPM COUNT: %d \r \n", rpm_tick_count);
  HAL_UART_Transmit(&huart6,txd_msg_buffer, strlen((char*)txd_msg_buffer), 1000);

    //RGB LED code
  HAL_GPIO_WritePin(GPIOB, BLU_Pin, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GRN_Pin, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, RED_Pin, GPIO_PIN_RESET);

  //ultra sonic sensor code
  hcsr04_Rx_flag = 0;
  first_edge = 0;
  time_edge1 = 0;
  time_edge2 = 0;
  time_diff = 0;
  distance = 0;

  HCSR04_TRIG_PULSE();

  while(hcsr04_Rx_flag == 0){};

  time_diff = time_edge2-time_edge1;
  distance = (int)(((float)time_diff)/54);
  sprintf((char*)txd_msg_buffer, "PULSE WIDTH is: %d \r\n", time_diff);
  HAL_UART_Transmit(&huart6, txd_msg_buffer, strlen((char*)txd_msg_buffer), 1000);

  sprintf((char*)txd_msg_buffer, "Distance is: %d \r\n", distance);
  HAL_UART_Transmit(&huart6, txd_msg_buffer, strlen((char*)txd_msg_buffer), 1000);
```

# DESIGN TEAMS

# UW BAJA SAE

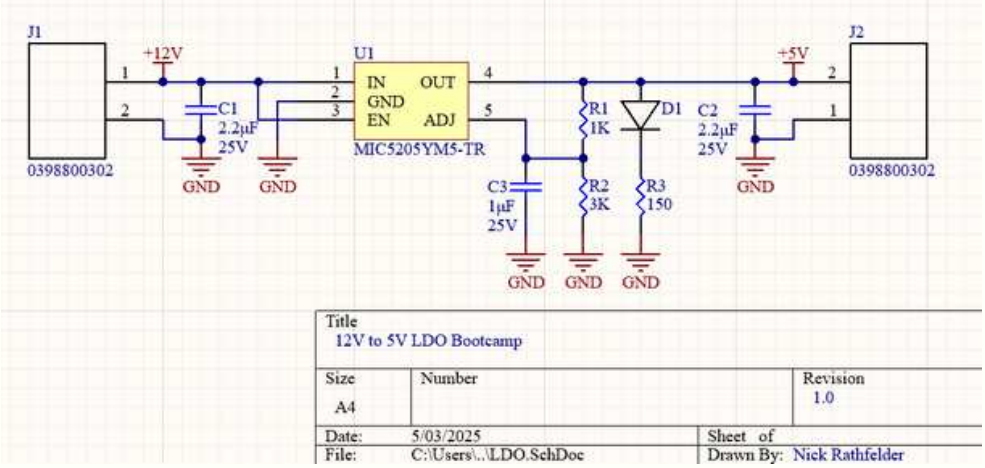
Fall 2024
Competition Car

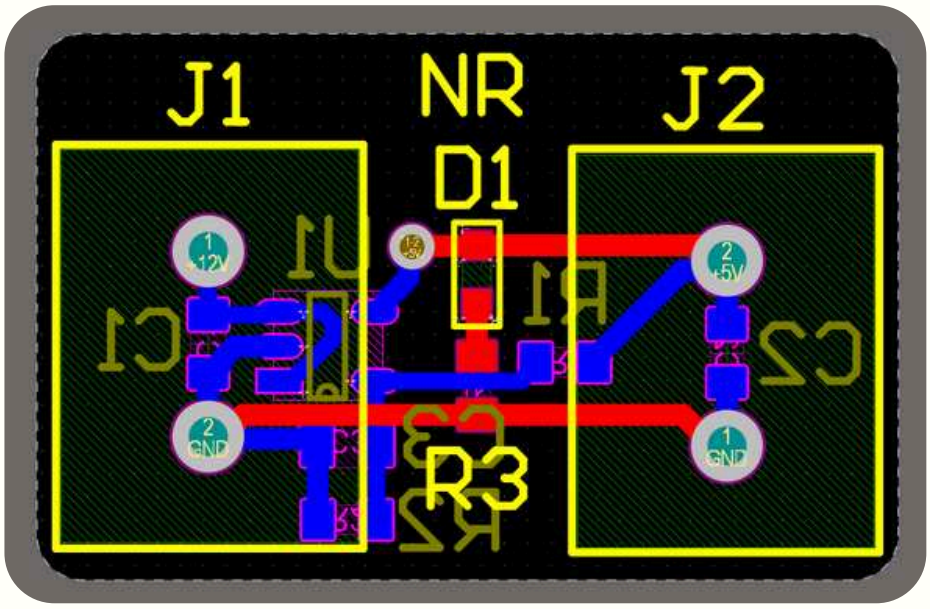

## Electrical Wire Harness Schematic



# UW WARG

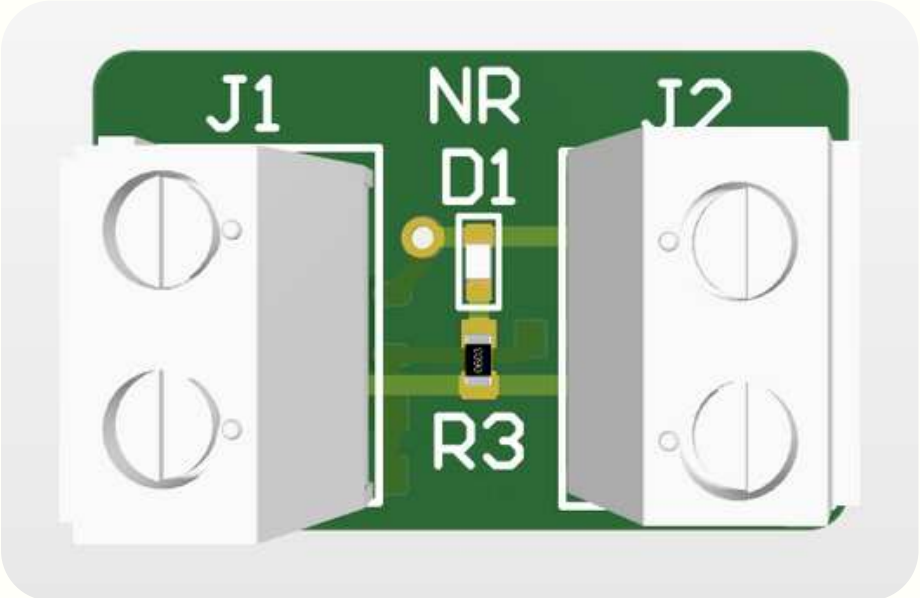
Bootcamp
Schematic




Bootcamp
Layout

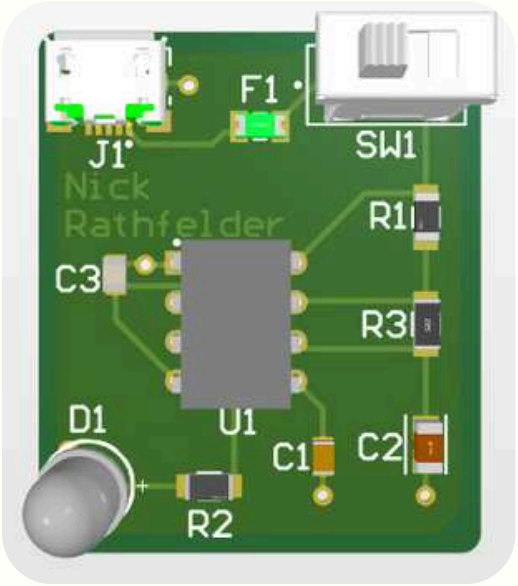
Bootcamp
3D Model

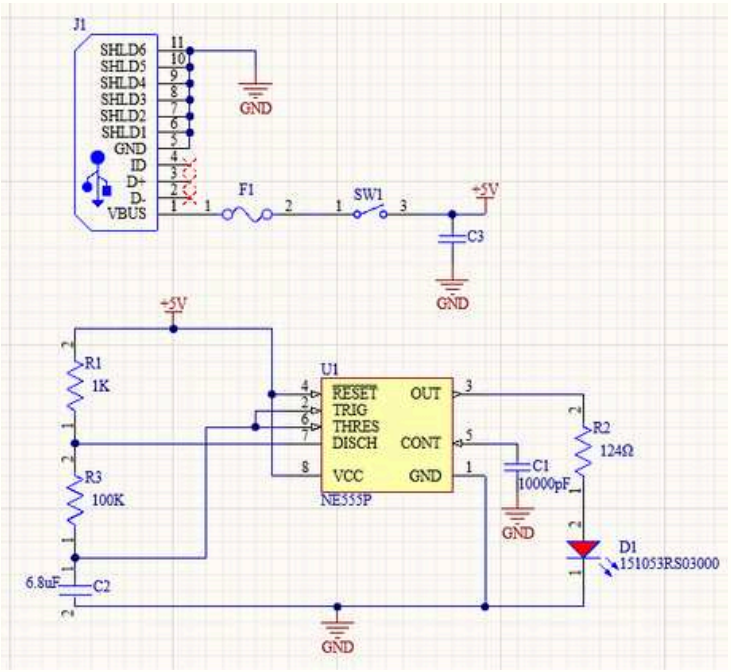


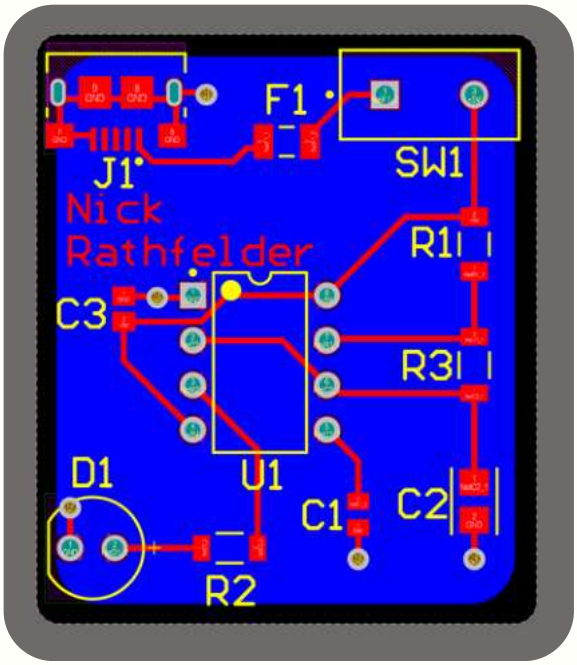
# MISCELLANEOUS

# PCB DESIGN 1

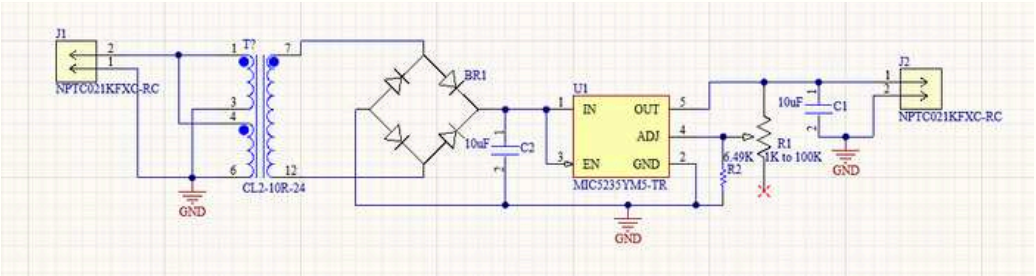
Blinking LED
Controller

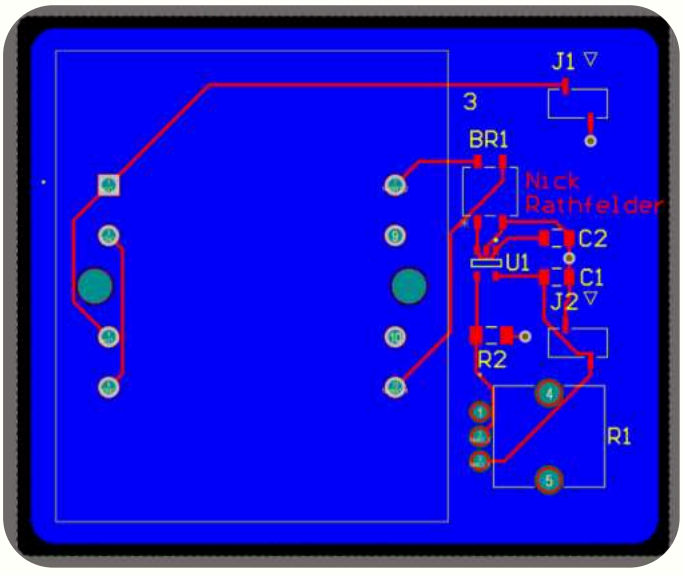


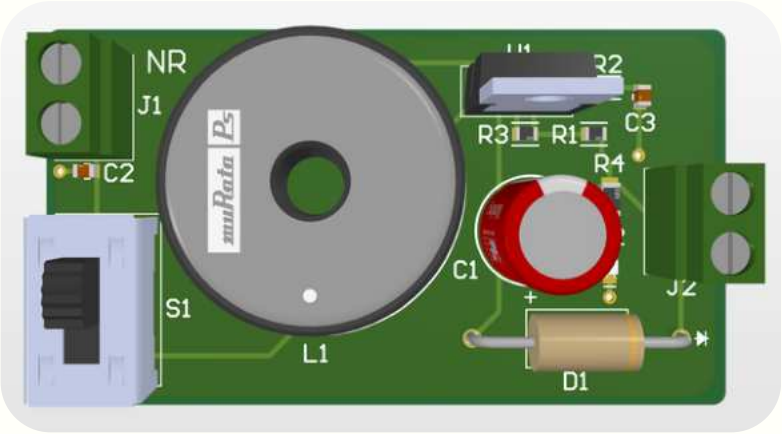




AC-DC
Power Supply

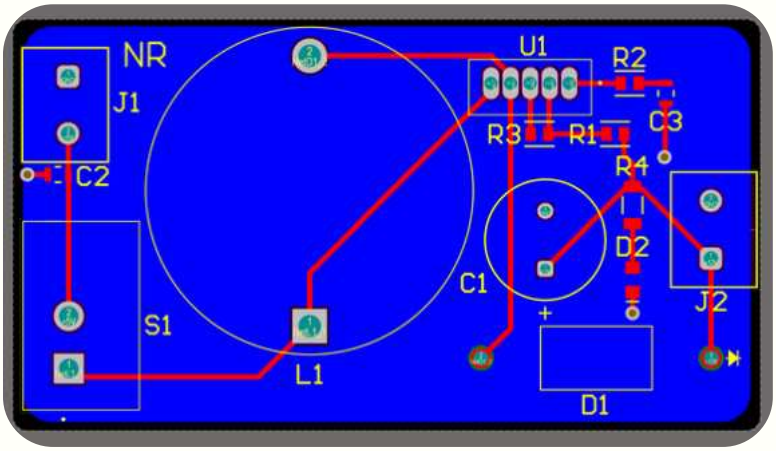


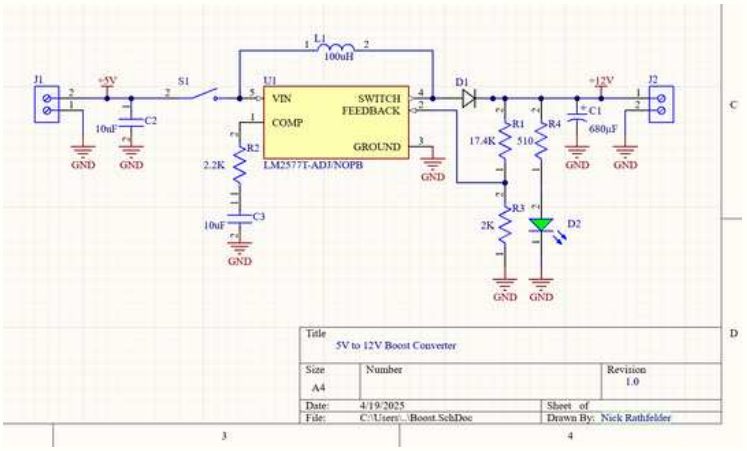




# PCB DESIGN 2

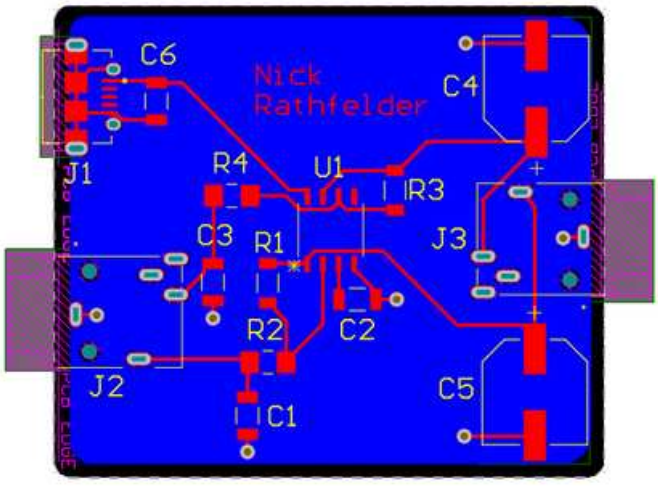


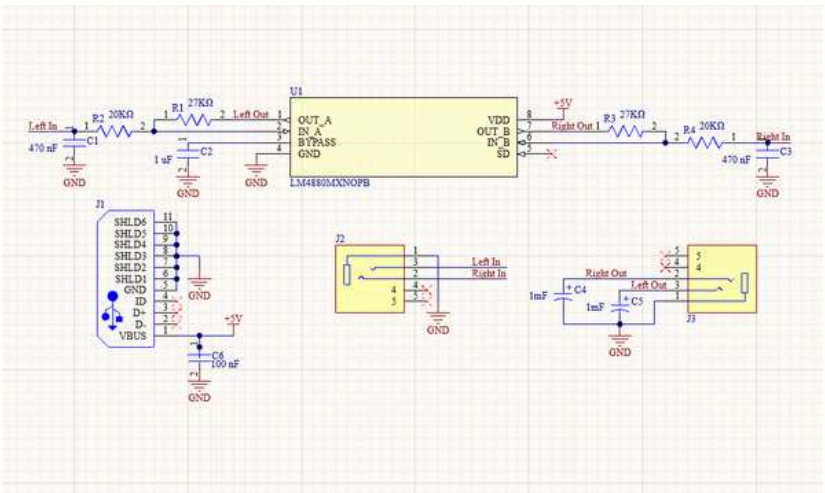

5V to 12V
Boost Converter

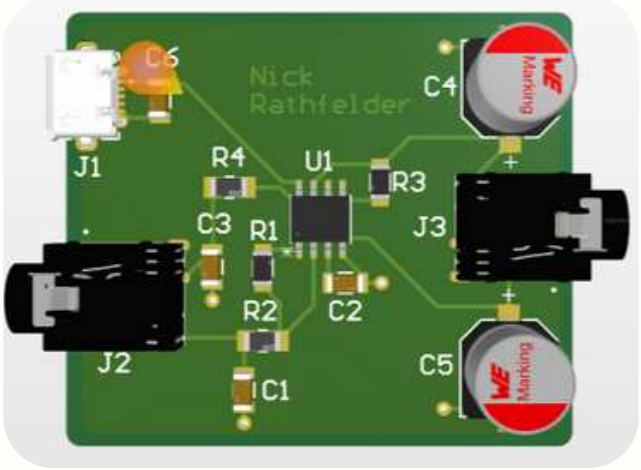



5W Audio
Amplifier

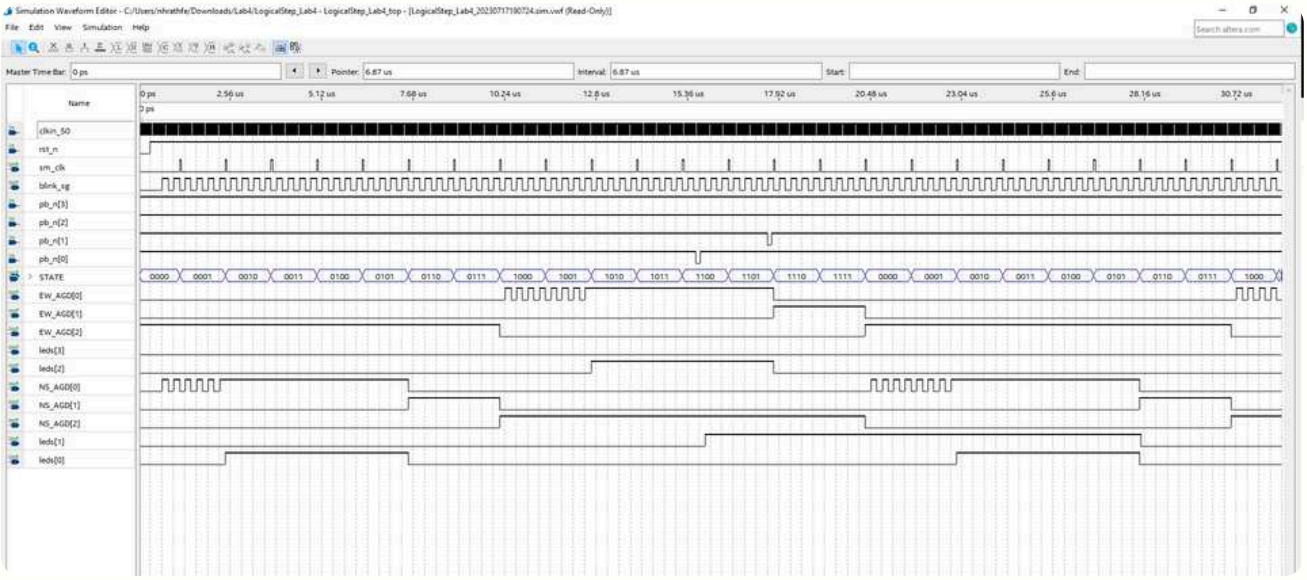


# TRAFFIC LIGHT CONTROLLER

## Output Waveform




## VHDL Top File

Date: July 19, 2023 LogicalStep_Lab4_top.vhd Project: LogicalStep_Lab4

```
72          register_clr          : in std_logic; -- Additional signal to clear the register
(from state machine)
73          din                   : in std_logic; -- Data to be held in the register
74          dout                  : out std_logic -- Data being held by the register
75    );
76    end component;
77
78    component PB_filters port (
79    clkin          : in std_logic;
80    rst_n          : in std_logic;
81    rst_n_filtered : out std_logic;
82    pb_n           : in  std_logic_vector (3 downto 0);
83    pb_n_filtered  : out std_logic_vector(3 downto 0)
84    );
85    end component;
86
87    component state_Machine Port
88    (
89        blink_sig,clk_input, reset, NS_request,EW_request     : IN std_logic; --Blink signal
for flashing LED control, 1hz clock input, synchronized reset button
90        --North/South and East/West pedestrian crossing requests (come from the holding
registers)
91        NS_AGD,EW_AGD                 : OUT std_logic_vector(2 downto 0);
92        -- 3 bit vectors that hold the values of which segement on the display to activate
(either segment A for the red light, G for the amber light, and D for the green light
93        NS_cross, EW_cross,NS_clr_reg,EW_clr_reg        : out std_logic;
94        -- 2 crosswalk outputs for when pedestrians can cross, 2 clr register outputs when a
cross request has been dealt with
95        State_led : out std_logic_vector(3 downto 0)
96        --4 bit vector containing the current state of the state machine
97    );
98    end component;
99    --
100
101   ----------------------------------------------------------------------------------------
--------
102      CONSTANT sim_mode                : boolean := FALSE; -- set to FALSE for LogicalStep
board downloads
103                                                          -- set to TRUE for SIMULATIONS
104
105      SIGNAL sm_clken, blink_sig       : std_logic; -- Clocks used to control the changing of
the state machine and the flashing green blinking signal
106
107      SIGNAL pb                        : std_logic_vector(3 downto 0); --pb(0) is used for
north/south crossing requests, pb(1) is used for east/west crossing requests
108      SIGNAL rst                       :std_logic; --Signal for unsynchronized, active high,
filtered reset
109
110      SIGNAL rst_n_f                   : std_logic; -- Filtered active low reset
111      SIGNAL pb_n_f                      : std_logic_vector(3 downto 0); --Filtered active
low push buttons
112      SIGNAL sync_rst                  :std_logic; -- Active high, filtered reset synchronized
with the clock
113      SIGNAL sreg:   std_logic_vector(1 downto 0); -- Intermediate signal used for holding the
EW and NS cross requests before being put into the registers
114
115      SIGNAL NS_controller             :std_logic_vector (2 downto 0); -- Holds the 3 bit NS
light output vector from the state machine
116      SIGNAL EW_controller             :std_logic_vector (2 downto 0); -- Holds the 3 bit EW
light output vector from the state machine
117      SIGNAL NS_seven_seg              :std_logic_vector (6 downto 0); -- Holds the 7 bit NS
seven segment input (concatenated version of NS_controller)
118      SIGNAL EW_seven_seg              :std_logic_vector (6 downto 0); -- Holds the 7 bit EW
seven segment input (concatenated version of EW_controller)
119
120      SIGNAL NS_cross_request                :std_logic; -- Signal that feeds the output from
the NS cross register to the state machine
121      SIGNAL EW_cross_request          :std_logic; -- Signal that feeds the output from
the EW cross register to the state machine
122
123      SIGNAL NS_clear                        :std_logic; -- Signal activated by the state
machine to clear the NS cross register when a request has been dealt with
124      SIGNAL EW_clear                        :std_logic; -- Signal activated by the state
machine to clear the EW cross register when a request has been dealt with
125
126   BEGIN
```





```
   ----------
   PB_Filter: PB_filters port map(clkin_50,rst_n,rst_n_f,pb_n,pb_n_f); --Filters all push
   button inputs and reset input
   PB_Invert: pb_inverters port map (rst_n_f,rst,pb_n_f,pb); -- Inverts all push button input
   and reset input to be active high
   Reset_Sync: synchronizer port map(clkin_50,sync_rst,rst,sync_rst); -- Synchronizes the
   reset button to eliminate metastability

   Clock_Gen: clock_generator     port map (sim_mode, sync_rst, clkin_50, sm_clken, blink_sig
   -- Takes the 50 mhz clock and turns it into the sm_clken for the state machine
   -- and the blink_sig for the flashing green light signal (1 hz for the sm_clken, 4hz for
   the blink_sig)

   EW_Sync: synchronizer port map(clkin_50,sync_rst,pb(1),sreg(1)); -- Synchronizes the
   east/west cross request with the clock (push button 1)
   NS_Sync: synchronizer port map(clkin_50,sync_rst,pb(0),sreg(0)); -- Synchronizes the
   north/south cross request with the clock (push button 0)

   EW_request: holding_register port map(clkin_50,sync_rst,EW_clear,sreg(1),EW_cross_request
   -- Holds the synced EW cross request for access by the state machine (reset by the sync
   reset or clear signal)
   NS_request: holding_register port map(clkin_50,sync_rst,NS_clear,sreg(0),NS_cross_request
   -- Holds the synced NS cross request for access by the state machine (reset by the sync
   reset or clear signal)

   State_Processing: state_Machine port map(blink_sig,sm_clken,sync_rst,NS_cross_request,
   EW_cross_request, NS_controller,EW_controller,leds(0),leds(2),NS_clear,EW_clear,leds (7
   downto 4));
   -- State machine takes in the blink_sig clock (for running the flashing green LED),
   sm_clken clock(for running the rest of the state machine), and both cross request signals
   from the registers
   -- Outputs the NS and EW_AGD light vectors (3 bit) for concatenation, the NS and EW
   crosswalk indicators to leds 0 and 2 respectively, and signals for clearing the 2 registers
   when a crossing request has been dealt with

   NS_seven_seg <= NS_controller(1) & "00" & NS_controller(0) & "00" & NS_controller(2); --
   Concatenates the 3 bit NS light output vector to a 7 bit signal for use in the seven
   segment display
   EW_seven_seg <= EW_controller(1) & "00" & EW_controller(0) & "00" & EW_controller(2); --
   Concatenates the 3 bit EW light output vector to a 7 bit signal for use in the seven
   segment display

   leds(3) <= EW_cross_request; -- Outputs whether there is an active EW cross request to led
   leds(1) <= NS_cross_request; -- Outputs whether there is an active NS cross request to led

   seven_seg_mux: segment7_mux port map(clkin_50,NS_seven_seg,EW_seven_seg,seg7_data,seg7_ch
   ,seg7_char1);
   -- Outputs the final light output to the seven segment displays using the concatenated 7
   bit light output vectors

   END SimpleCircuit;
```