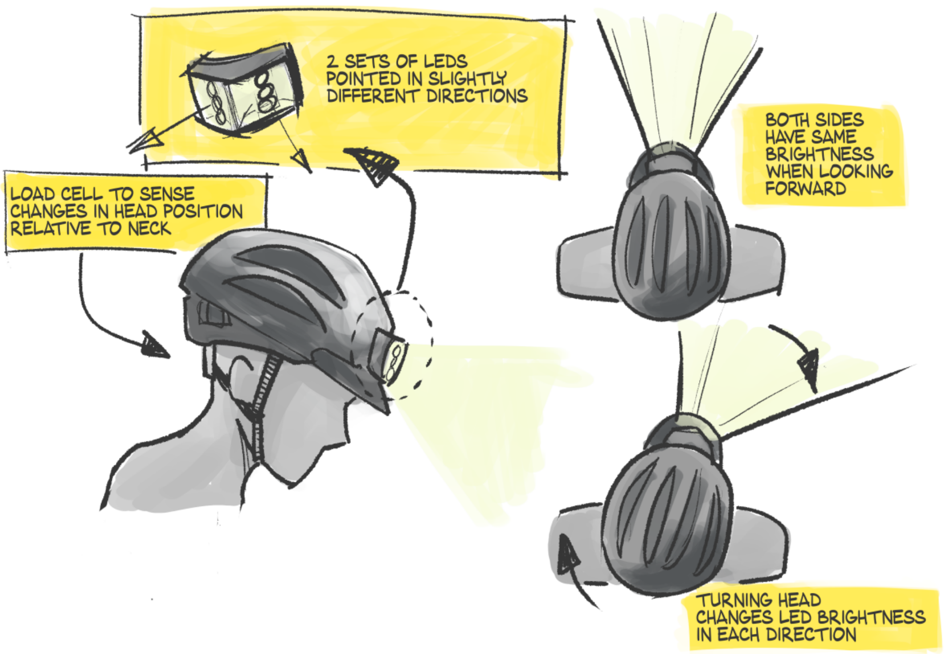
Arduino Shield HeadLight

Edward Manson (75953196)
Alex Stiles (18120890)

August 27, 2021



Contents

List of Figures

List of Tables

Listings

1 Introduction

2 Background

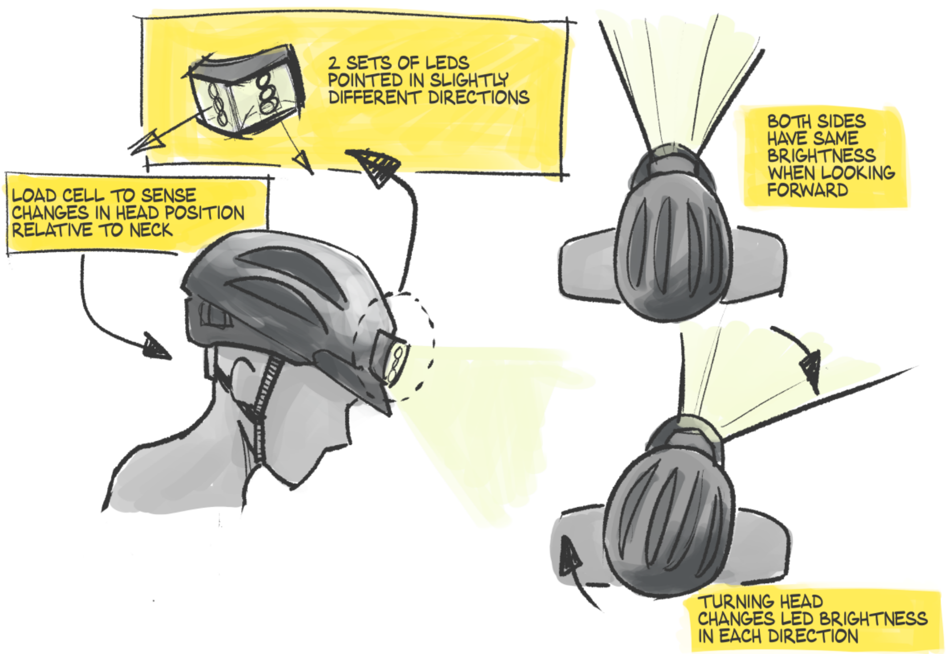


Figure 1: A sketch of the HeadLight

3 Requirements & Specifications

3.1 Requirements

3.2 Specifications

4 Design Overview & Rationale

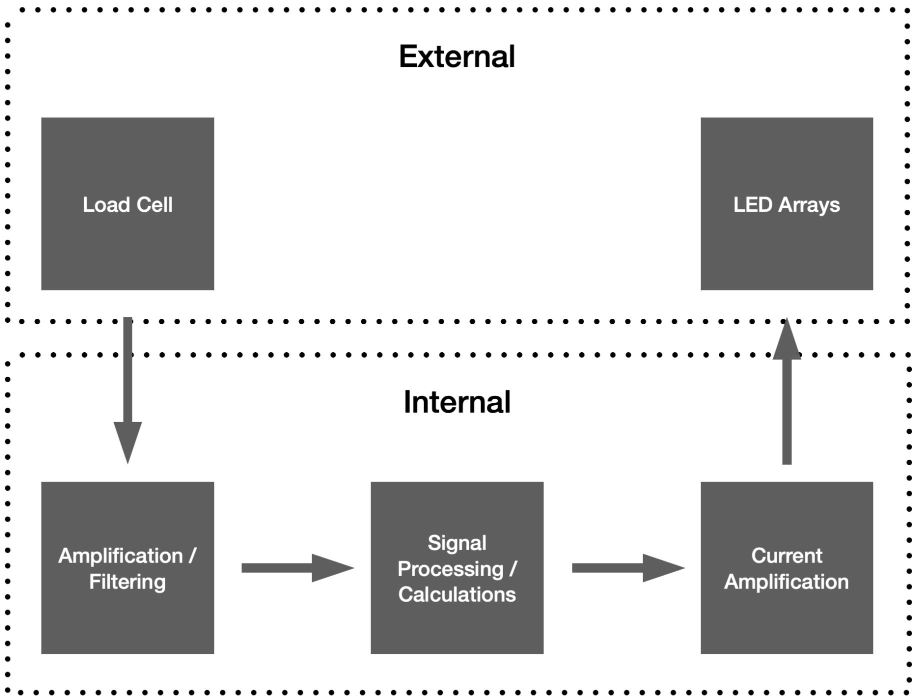


Figure 2: A block diagram of the function of the HeadLight

4.1 Arduino Shield

A load cell is used to measure the force of the head rotation, which has an internal Wheatstone bridge. The output from the Wheatstone bridge goes into an instrumentation amplifier. The instrumentation amplifier consists of non-inverting amplifiers on the output of the Wheatstone bridge, which then goes into a differential amplifier, as shown in figure 3. The output of the differential amplifier goes into an LC low pass filter to remove high-frequency noise.

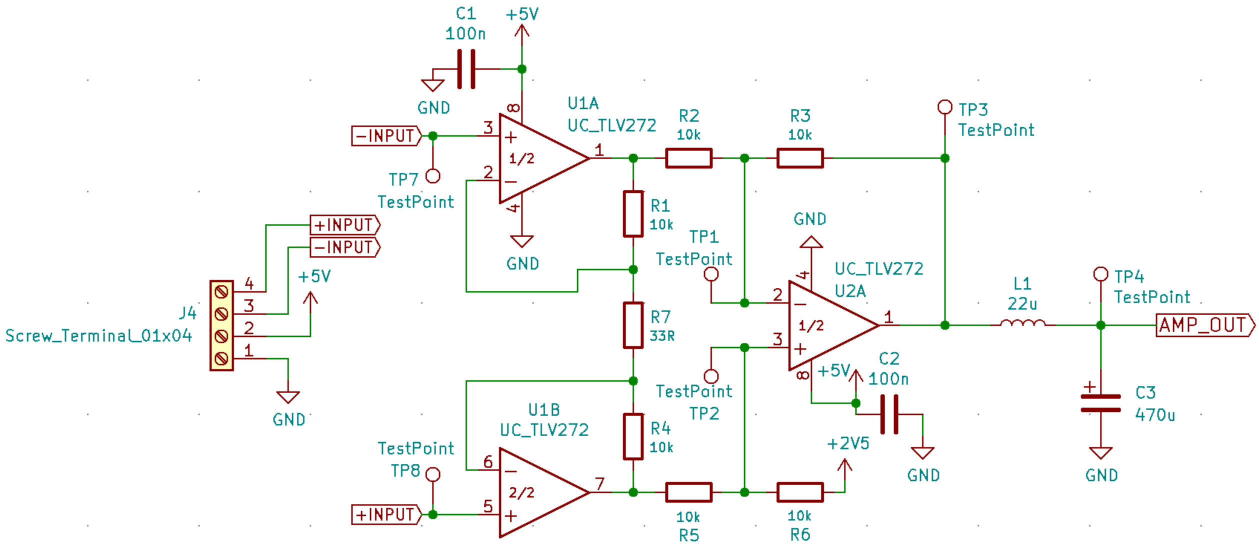


Figure 3: The layout of the instrumentation amplifier.

The output of the differential amplifier can end up being a negative value, as the Arduino can not read negative voltage, and there is no negative supply for the differential amplifier; the differential amplifier has to be biased. A voltage reference was constructed out of a voltage divider of equal values and a buffer to bias the instrumentation amplifier (shown in figure 4).

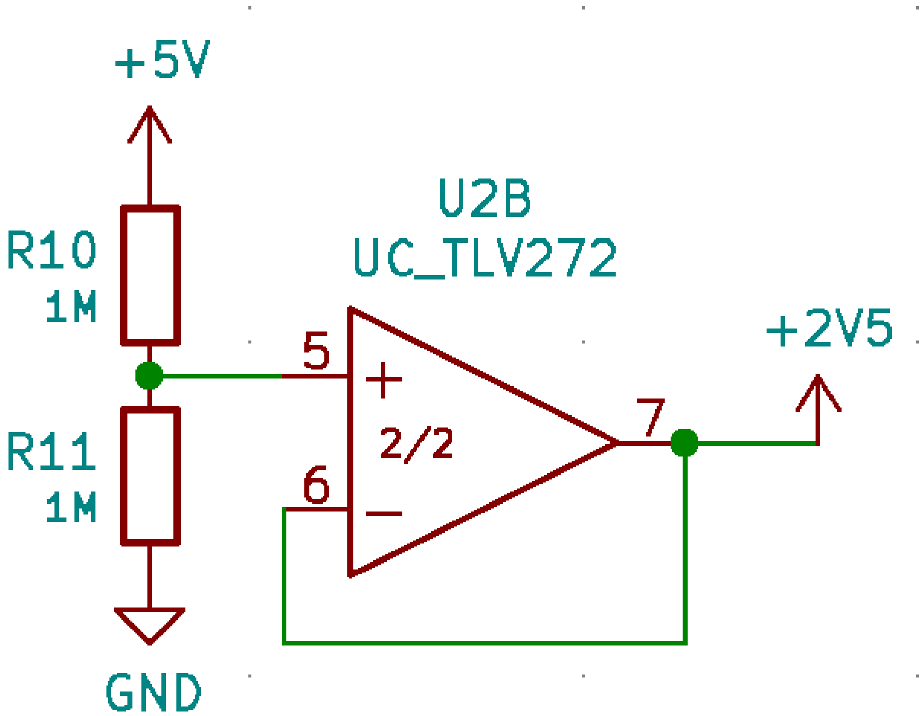


Figure 4: The layout of the voltage reference.

For each LED array, the PWM output from the Arduino goes into the gate of a MOSFET. The current limiting resistor is connected to the anode of the LEDs, and the cathode of the LEDs is connected to the drain of the MOSFET.

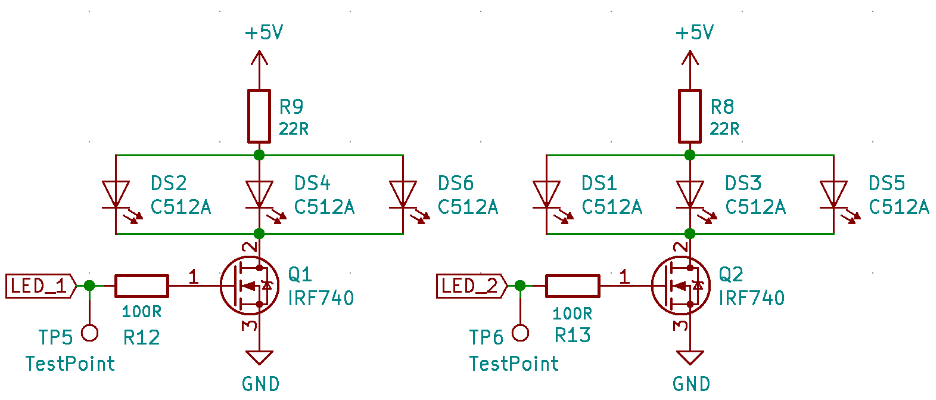


Figure 5: The layout of the LED arrays.

4.1.1 PCB Layout

The PCB was laid out to ensure that the ground plane would have no cuts, maximising EMC performance. Decoupling capacitors were placed right next to the power pins of the ICs. Additional care was taken so that traces were not too close to any antipads, as having traces too close to antipads can worsen EMC **emc**. Test points were added so that we can troubleshoot any problems when assembling the prototype. 1x1 header were used as ground test points so that the ground clip of the oscilloscope probe can be attached.

4.1.2 Amplifier

An instrumentation amplifier was chosen to amplify the signal from the load cell. The instrumentation amplifier consists of two non-inverting amplifiers, of which the outputs go into a differential amplifier. As the Arduino cannot read negative voltages, nor is there a negative supply, the differential amplifier was biased with 2.5 V. This results in the positive difference being between 2.5 V and 5 V, and the negative difference is between 0 V and 2.5 V. A one op-amp differential amplifier has an extremely low input impedance; thus the Wheatstone bridge must have a matched impedance. A two op-amp instrumentation amplifier would require one less package but would be less stable than the three op-amp instrumentation amplifier. The three op-amp instrumentation amplifier also has a spare op-amp which is then utilised for the voltage reference.

An initial gain of 24 Ω was calculated (as shown in equation (4)) **ti:sloa034**, but after some test on a breadboard, the gain was changed 33 Ω. This was partly due to the load cell to measure ~50 % more than rated **htc:tal221**.

$$V_O = \left[(\text{Sig}^+) - (\text{Sig}^-)\right]\left[\frac{R_4}{R_2}\left[\frac{2R_f}{R_g}+1\right]\right] \tag{1}$$

$$\begin{aligned} R_g &= \frac{2R_f}{\left[\dfrac{V_O}{(\text{Sig}^+) - (\text{Sig}^-)}\dfrac{R_2}{R_4}\right]-1} && (2)\\ &= \frac{2\cdot 10\,\text{k}\Omega}{\left[\dfrac{2.5\,\text{V}}{3\,\text{mV}}\dfrac{10\,\text{k}\Omega}{10\,\text{k}\Omega}\right]-1} && (3)\\ &= 24.03\,\Omega && (4)\end{aligned}$$

4.1.3 Voltage reference

A voltage divider was used to produce 2.5 V to bias the differential amplifier. It was decided to use a buffer on the output of the voltage divider so that the common-mode rejection error is minimised. This is due to the op-amp's low output impedance, which does not introduce any noticeable common-mode rejection error. A 2.5 V Zener diode would have been preferred. The resistors in the voltage divider have tolerances and therefore may not give exactly 2.5 V; however, 2.5 V Zener diodes were not available from the store. 1 MΩ resistors were chosen to limit the current and thus power consumption.

4.1.4 Filter

A filter was used to filter out high-frequency noise from the load cell. The filter was chosen to be an LC filter. An RC filter could have been used, but it would have caused the signal's voltage to drop slightly due to the voltage drop across the resistor. The LC filter does require more space on the PCB, but the PCB had enough room for the capacitor and the inductor. LC filters also have better attenuation as the attenuation slope is twice as steep as an RC filter. The filter was placed on the output of the differential amplifier. If filters were placed on the inputs of the differential op-amp, more components would be needed, and there would not be enough space on the board. A 22 µH inductor was chosen due to its availability in the electronics store. A 470 µF capacitor was selected as a compromise of its physical size, cutoff frequency, and availability. The cutoff frequency was found to be 1.595 kHz, as shown below:

$$
\begin{aligned}
f &= \frac{1}{2\pi\sqrt{LC}} &(5)\\
&= \frac{1}{2\pi\sqrt{22\,\mu\text{H}470\,\mu\text{F}}} &(6)\\
&= 1.595\,\text{kHz} &(7)
\end{aligned}
$$

4.1.5 LED arrays

Each LED array has a MOSFET; this was done to have enough current supplied to the LEDs. The pins that can output PWM are only able to supply 40 mA, so the outputs from these pins were connected to the gates of the MOSFETs. This allows the LEDs to take full current and are thus brighter. IRF740 MOSFETs were chosen due to their availability in the store and their low resistance. They have a max current of 10 A, so more LEDs can be added if required. The low resistance of the IRF740 (0.55 Ω), reduces the power losses when the MOSFET is on. Individual resistors could have been matched to each LED; however, this would increase the number of components and take up extra space on the PCB. 100 Ω resistors were placed between the PWM output from the Arduino and the gate of the MOSFETs, this was done to minimise ringing and to limit current.

5 Testing

The PCB will be assembled and attached to an Arduino. The code will be uploaded to the Arduino. The load cell will be manipulated, and the PWM signal will be probed using an oscilloscope. The duty cycle of the PWM signal will be compared to the expected duty cycle. The LEDs will then be connected, and the load cell manipulated to check that the LEDs behave as expected in terms of brightness and direction changes. The load cell signal will be probed to ensure that the gain resistor is the correct value. This will be done by placing an appropriately 100 g mass on the load cell and measuring the voltage that is output by the instrumentation amplifier.

6 Conclusions

Appendices

A Applications of the Design Toolkit

Table 1: Applications of the Design Toolkit

Tool Type	Tool Name	Where/How Used
Strategy	Divide & Conquer	Divide and conquer was used to split up the hardware design into: ▪ Instrumentation amplifier ▪ LED arrays and driver ▪ 2.5 V reference ▪ LC filter The project itself was also divided up so that one person did the software, and the other person did the hardware.
Design Principle	PCB layout	The layout of the PCB was done with the aim of complying with the PCB design rules, e.g. Having components neatly lined up, minimising acute angles of tracks, increasing clearance between components and tracks, and having high current tracks where needed.
Design Principle	Schematic layout	The schematic was done with the aim to follow the layout guidelines, for example, using labels, using power ports, separating major parts of the circuit, and ensuring that the labels are clear and easy to read.
Design Principle / Strategy	Modularity	The code was written with the aim to be modular, e.g. multiple header files and task specific files.
Design Principle	Testability	Test points were added around the circuit to aid in easy testing of the circuit's function.
Strategy	Get another perspective	Both the PCB and the schematic were checked by people other than the designer, such as the other group member and people external to the group.
Building Block	Amplifier	An instrumentation amplifier was constructed out of operational amplifiers to appropriately amplify the signal from the Wheatstone bridge in the load cell.
Building Block	Filter	An LC filter was used to filter out noise from the load cell, and to not attenuate the signal.
Building Block	Buffer	A buffer was used to create a low impedance voltage reference. Buffers were also used on the output of the Wheatstone bridge to provide a high impedance for the Wheatstone bridge so that the signal was not excessively attenuated.
Building Block	Current Limiting Resistor	Current limiting resistors were usd to limit the gate current for the MOSFETs and the current through the LEDs.
Building Block	Arduino microcontroller	The Arduino is the main controller for the circuit, it controls the brightness and reads the load cell.

continues on next page

Table 1 – continued from previous page

Tool Type	Tool Name	Where/How Used
Building Block	Pull up resistor	An internal pull up resistor is used for the button. A pull up was chosen so that the internal pull up resistor could be utilised, keeping the parts count down.
Building Block	Analog input	An analog input was used to read the amplified signal from the Wheatstone bridge of the load cell.
Building Block	Digital output	Digital outputs were used for the MOSFETs that control the LEDs, amd for the auxillary output. Digital outputs are capable of outputting PWM.

B Code

```c
#include <stdint.h>
#include <stdbool.h>

#include "button.h"

Button_t init_button(uint8_t pin_number, void (*callbackFunction)()) {
    Button_t button = {
        .pin_number = pin_number,
        .debounce_delay = DEBOUNCE_DELAY,
        .last_trigger_time = 0,
        .state = true,
        .last_state = true,
        .callbackFunction = callbackFunction
    };

    return button;
}

void button_clicked(Button_t* button) {
    button->callbackFunction();
}

void debounce_state(Button_t* button, bool read_button_state, long unsigned current_time)
     {
    if (read_button_state != button->last_state) {
        button->last_trigger_time = current_time;
    }

    if ((current_time - button->last_trigger_time) > button->debounce_delay) {
        if (read_button_state != button->state) {
            button->state = read_button_state;
            if (button->state) {
                button_clicked(button);
            }
        }
    }

    button->last_state = read_button_state;
}
```

Listing 1: Button code.

```c
/**
 * @file button.h
 * @author Alex Stiles
 * @brief Provides the types and function prototypes required for the button.
 * @version 0.1 alpha
 * @date 2021-08-15
 *
 * @copyright Copyright (c) 2021
 *
 */

#ifndef BUTTON_H
#define BUTTON_H

#if defined(__cplusplus)
extern "C" {
#endif

#include <stdint.h>
#include <stdbool.h>

#define DEBOUNCE_DELAY 20 // Debounce delay in ms
```

```c

typedef struct Button_s Button_t;

struct Button_s {
    /* General */
    uint8_t pin_number;

    /* Debounce */
    long unsigned debounce_delay;
    long unsigned last_trigger_time;

    bool state;
    bool last_state;

    /* Callback */
    void (*callbackFunction)();
};

Button_t init_button(uint8_t pin_number, void (*callbackFunction)());
void buttonClicked(Button_t* button);
void debounce_state(Button_t* button, bool read_button_state, long unsigned current_time)
    ;

#if defined(__cplusplus)
}
#endif

#endif // BUTTON_H
```

Listing 2: Button header code.

```c
/**
 * @file LED_Group.c
 * @author Alex Stiles
 * @brief Contains the source code for a LED group.
 * @version 0.1
 * @date 2021-08-13
 *
 * @copyright Copyright (c) 2021
 *
 * This source file provides the functionality to get and set the group's brightness (
     from 0 to 255) as required by the Arduino's PWM output.
 * It also provides getters for the group pin number and direction.
 *
 */

#include <stdint.h>

#include "LED_Group.h"

/**
 * @brief Get the group brightness
 *
 * @param led_group
 * @return int8_t
 */
uint8_t get_group_brightness(const LED_Group_t* const led_group) {
    return led_group->group_brightness;
}

/**
 * @brief Set the group brightness
 *
 * @param led_group
 * @param group_brightness
 */
void set_group_brightness(LED_Group_t* const led_group, uint8_t group_brightness) {
```

```
    led_group->group_brightness = group_brightness;
}

/**
 * @brief Get the group pin number
 *
 * @param led_group
 * @return int8_t
 */
uint8_t get_group_pin_number(const LED_Group_t* const led_group) {
    return led_group->group_pin_number;
}

/**
 * @brief Get the group direction
 *
 * @param led_group
 * @return Direction_t
 */
Direction_t get_group_direction(const LED_Group_t* const led_group) {
    return led_group->direction;
}
```

Listing 3: LED group code.

```
k ai/**
 * @file LED_Group.h
 * @author Alex Stiles
 * @brief Provides the types and function prototypes required for the LED groups.
 * @version 0.1 alpha
 * @date 2021-08-13
 *
 * @copyright Copyright (c) 2021
 *
 */

#ifndef LED_GROUP_H
#define LED_GROUP_H

#if defined(__cplusplus)
extern "C" {
#endif

#include <stdint.h>

#include "voltage.h"

typedef enum {
    LEFT = 0,
    RIGHT
} Direction_t;

typedef struct {
    uint8_t group_pin_number;
    Direction_t direction;
    uint8_t group_brightness;
} LED_Group_t;

uint8_t get_group_brightness(const LED_Group_t* const led_group);
void set_group_brightness(LED_Group_t* const led_group, uint8_t group_brightness);

uint8_t get_group_pin_number(const LED_Group_t* const led_group);

Direction_t get_group_direction(const LED_Group_t* const led_group);

#if defined(__cplusplus)
}
```

```c
#endif

#endif // LED_GROUP_H
```

Listing 4: LED group header code.

```c
/**
 * @file LED_Group_Collection.c
 * @author Alex Stiles
 * @brief Contains the source code for manipulating all the induvidual LED groups (see
     LED_Group.c)
 * @version 0.1 alpha
 * @date 2021-08-13
 *
 * @copyright Copyright (c) 2021
 *
 * This source file provides the functionality to add and get induvidual LED groups.
 * It also provides the ability to update the required PWM output to each of the
     induvidual gruops, given a load voltage.
 *
 */

#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "LED_Group_Collection.h"

/**
 * @brief Create a LED group collection, and add an LED group to each direction (LEFT and
      RIGHT) with the pins set to left_pin_number and right_pin_number respectively.
 *
 * @param left_pin_number
 * @param right_pin_number
 * @return LED_Group_Collection_t
 */
LED_Group_Collection_t init_led_group_collection(uint8_t left_pin_number, uint8_t
    right_pin_number) {
    LED_Group_Collection_t led_group_collection = {.led_groups = {0}, .mode = FOLLOW, .
        mode_changed = false};

    LED_Group_t led_group_one = {
        .group_pin_number = left_pin_number,
        .direction = LEFT,
        .group_brightness = 0
    };

    LED_Group_t led_group_two = {
        .group_pin_number = right_pin_number,
        .direction = RIGHT,
        .group_brightness = 0
    };

    set_led_group(&led_group_collection, led_group_one, LEFT);
    set_led_group(&led_group_collection, led_group_two, RIGHT);

    return led_group_collection;
}

/**
 * @brief Get a led group
 *
 * @param led_group_collection
 * @param direction
 * @return LED_Group_t*
 */
```

```
LED_Group_t* get_led_group(LED_Group_Collection_t* led_group_collection, Direction_t
    direction) {
    return &(led_group_collection->led_groups[direction]);
}

/**
 * @brief Set a direction's led group
 *
 * @param led_group_collection
 * @param led_group
 * @param direction
 */
void set_led_group(LED_Group_Collection_t* led_group_collection, LED_Group_t led_group,
    Direction_t direction) {
    led_group_collection->led_groups[direction] = led_group;
}

/**
 * @brief Set a led group brightnesses
 *
 * @param led_group_collection
 */
void set_led_group_brightnesses(LED_Group_Collection_t* led_group_collection, Load_Cell_t
    * load_cell) {

    led_group_collection->mode_changed = false;

    uint8_t group_one_brightness;
    uint8_t group_two_brightness;

    if (led_group_collection->mode == FOLLOW) {
        group_one_brightness = (uint8_t) round((map_voltage_to_range(load_cell->voltage,
            load_cell->max_voltage, 1) + 0.5) * ARDUINO_PWM_WRITE_RANGE);
        group_two_brightness = (ARDUINO_PWM_WRITE_RANGE - group_one_brightness);
    } else if (led_group_collection->mode == FULL) {
        group_one_brightness = ARDUINO_PWM_WRITE_RANGE;
        group_two_brightness = ARDUINO_PWM_WRITE_RANGE;
    }

    set_group_brightness(get_led_group(led_group_collection, LEFT), group_one_brightness)
        ;
    set_group_brightness(get_led_group(led_group_collection, RIGHT), group_two_brightness
        );
}
```

Listing 5: LED group collection code.

```
/**
 * @file LED_Group_Collection.h
 * @author Alex Stiles
 * @brief Provides the types and function prototypes required for the collection of LED
   groups.
 * @version 0.1
 * @date 2021-08-13
 *
 * @copyright Copyright (c) 2021
 *
 */

#ifndef LED_GROUP_COLLECTION_H
#define LED_GROUP_COLLECTION_H

#if defined(__cplusplus)
extern "C" {
#endif

#include <stdint.h>
```

```c
#include <stdbool.h>

#include "voltage.h"

#include "Load_Cell.h"
#include "LED_Group.h"

#define NUM_MODES 2

typedef enum {
    FOLLOW = 0,
    FULL
} Light_Mode_t;

typedef struct {
    LED_Group_t led_groups[2];
    Light_Mode_t mode;
    bool mode_changed;
} LED_Group_Collection_t;

LED_Group_Collection_t init_led_group_collection(uint8_t left_pin_number, uint8_t
    right_pin_number);
LED_Group_t* get_led_group(LED_Group_Collection_t* led_group_collection, Direction_t
    direction);
void set_led_group(LED_Group_Collection_t* led_group_collection, LED_Group_t led_group,
    Direction_t direction);
void set_led_group_brightnesses(LED_Group_Collection_t* led_group_collectionm,
    Load_Cell_t* load_cell);

#if defined(__cplusplus)
}
#endif

#endif // LED_GROUP_COLLECTION_H
```

Listing 6: LED group collection header code.

```c
/**
 * @file Load_Cell.c
 * @author Alex Stiles
 * @brief Contains the source code for the TAL221 load cell.
 * @version 0.1 alpha
 * @date 2021-08-13
 *
 * @copyright Copyright (c) 2021
 *
 * This source file provides the functionality to poll the load cell.
 * Polling the load cell updates the voltage and dependant variables if required.
 *
 */

#include <stdint.h>
#include <stdbool.h>
#include <math.h>

#include <stdio.h>

#include "Load_Cell.h"

/**
 * @brief Create a load cell, with the pin set to LOAD_CELL_ANALOG_PIN and the deviation
   voltage breakpoint to DEVIATION_VOLTAGE_BREAKPOINT
 *
 * @return Load_Cell_t
 */
Load_Cell_t init_load_cell(uint8_t load_cell_pin_number) {
    Load_Cell_t load_cell = {
```

```
        .voltage = COMMON_MODE_VOLTAGE,
        .strain = 0,
        .angle = 0,
        .pin_number = load_cell_pin_number,
        .deviation_voltage_breakpoint = DEVIATION_VOLTAGE_BREAKPOINT,
        .max_voltage = ARDUINO_MAX_VOLTAGE,
        .strain_range = STRAIN_RANGE,
        .angle_range = ANGLE_RANGE
    };


    return load_cell;
}

/**
 * @brief Poll the load cell (by reading the voltage input at the LOAD_CELL_ANALOG_PIN) and compare it to the currently recorded voltage.
 * The methodology for detecting noise is rudimentary, lacking any requirement for the voltage to remain in a constrained range for a given time, reminiscent of button debouncing.
 * A more straightforward method is utilised due to hardware filtering using an active low pass filter. See shield schematic for details.
 *
 * @param load_cell pointer to Load_Cell_t struct
 * @return Whether a voltage deviation was read
 */
bool poll_sensor(Load_Cell_t* load_cell, float read_voltage) {
    bool deviation = false;

    // Compare to the current voltage to see if deviation detected

    if (fabsf(load_cell->voltage - read_voltage) > load_cell->deviation_voltage_breakpoint) {
        // Update the recorded voltage of the load cell
        load_cell->voltage = read_voltage;
        // Update the strain
        load_cell->strain = map_voltage_to_range(load_cell->voltage, load_cell->max_voltage, load_cell->strain_range);
        // Update the angle
        load_cell->angle = map_voltage_to_range(load_cell->voltage, load_cell->max_voltage, load_cell->angle_range);
        deviation = true;
    }

    return deviation;
}
```

Listing 7: Load cell code.

```
/**
 * @file Load_Cell.h
 * @author Alex Stiles
 * @brief Provides the types and function prototypes required for the TAL221 load cell.
 * @version 0.1 alpha
 * @date 2021-08-13
 *
 * @copyright Copyright (c) 2021
 *
 */

#ifndef LOAD_CELL_H
#define LOAD_CELL_H

#if defined(__cplusplus)
extern "C" {
#endif

```

```c
#include <stdint.h>
#include <stdbool.h>

#include "voltage.h"

#define STRAIN_RANGE 200
#define ANGLE_RANGE 180
#define DEVIATION_VOLTAGE_BREAKPOINT 0.1

typedef struct {
    // State of load cell at given time
    float voltage;
    int8_t strain;
    int8_t angle;

    // Specific details of load cell
    uint8_t pin_number;
    float deviation_voltage_breakpoint;
    float max_voltage;
    uint8_t strain_range;
    uint8_t angle_range;
} Load_Cell_t;

Load_Cell_t init_load_cell(uint8_t load_cell_pin_number);
bool poll_sensor(Load_Cell_t* load_cell, float read_voltage);

#if defined(__cplusplus)
}
#endif

#endif // LOAD_CELL_H
```

Listing 8: Load cell header code.

```c
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>

#include "voltage.h"

#include "LED_Group_Collection.h"
#include "LED_Group.h"
#include "Load_Cell.h"
#include "button.h"

#define LOAD_CELL_ANALOG_PIN 0

#define LEFT_GROUP_PIN 5
#define RIGHT_GROUP_PIN 6

#define BUTTON_PIN 7

#if !defined(ARDUINO)

LED_Group_Collection_t led_group_collection;
Load_Cell_t load_cell;
Button_t button;

void callbackFunction() {
    led_group_collection.mode = ++led_group_collection.mode % NUM_MODES;
}

int main(void) {
    led_group_collection = init_led_group_collection(LEFT_GROUP_PIN, RIGHT_GROUP_PIN);
    load_cell = init_load_cell(LOAD_CELL_ANALOG_PIN);
    button = init_button(BUTTON_PIN, &callbackFunction);
```

```c

    long unsigned current_time = 0;
    bool button_state = 0;

    debounce_state(&button, button_state, current_time);

    if (poll_sensor(&load_cell, 3.75)) {
        set_led_group_brightnesses(&led_group_collection, &load_cell);

        printf("Load Cell Voltage: %.1f/5 V\n", load_cell.voltage);
        printf("Left Group Brightness: %d/255 | Right Group Brightness: %d/255\n",
            get_group_brightness(get_led_group(&led_group_collection, LEFT)),
            get_group_brightness(get_led_group(&led_group_collection, RIGHT)));
        printf("Load Cell Strain: %d grams\n", load_cell.strain);
        printf("Head Angle: %d deg\n", load_cell.angle);
    }

   return EXIT_SUCCESS;
}

#endif
```

Listing 9: Main code.

```c
/**
 * @file voltage.c
 * @author Alex Stiles
 * @brief Provides helper functions when dealing with Arduino voltage
 * @version 0.1
 * @date 2021-08-13
 *
 * @copyright Copyright (c) 2021
 *
 */

#include "math.h"

#include "voltage.h"

float map_voltage_to_range(float voltage, float max_voltage, float range) {
    return ((voltage - 2.5) / max_voltage) * range;
}
```

Listing 10: Voltage ADC code.

```c
/**
 * @file voltage.h
 * @author Alex Stiles
 * @brief  Contains definations regarding the max voltage provided by the arduino and the common mode.
 * @version 1
 * @date 2021-08-13
 *
 * @copyright Copyright (c) 2021
 *
 */

#ifndef VOLTAGE_H
#define VOLTAGE_H

#define ARDUINO_MAX_VOLTAGE 5
#define ARDUINO_PWM_READ_RANGE 1023
#define ARDUINO_PWM_WRITE_RANGE 255
#define COMMON_MODE_VOLTAGE 2.5

float map_voltage_to_range(float voltage, float max_voltage, float range);

```

```
#endif // VOLTAGE_H
```

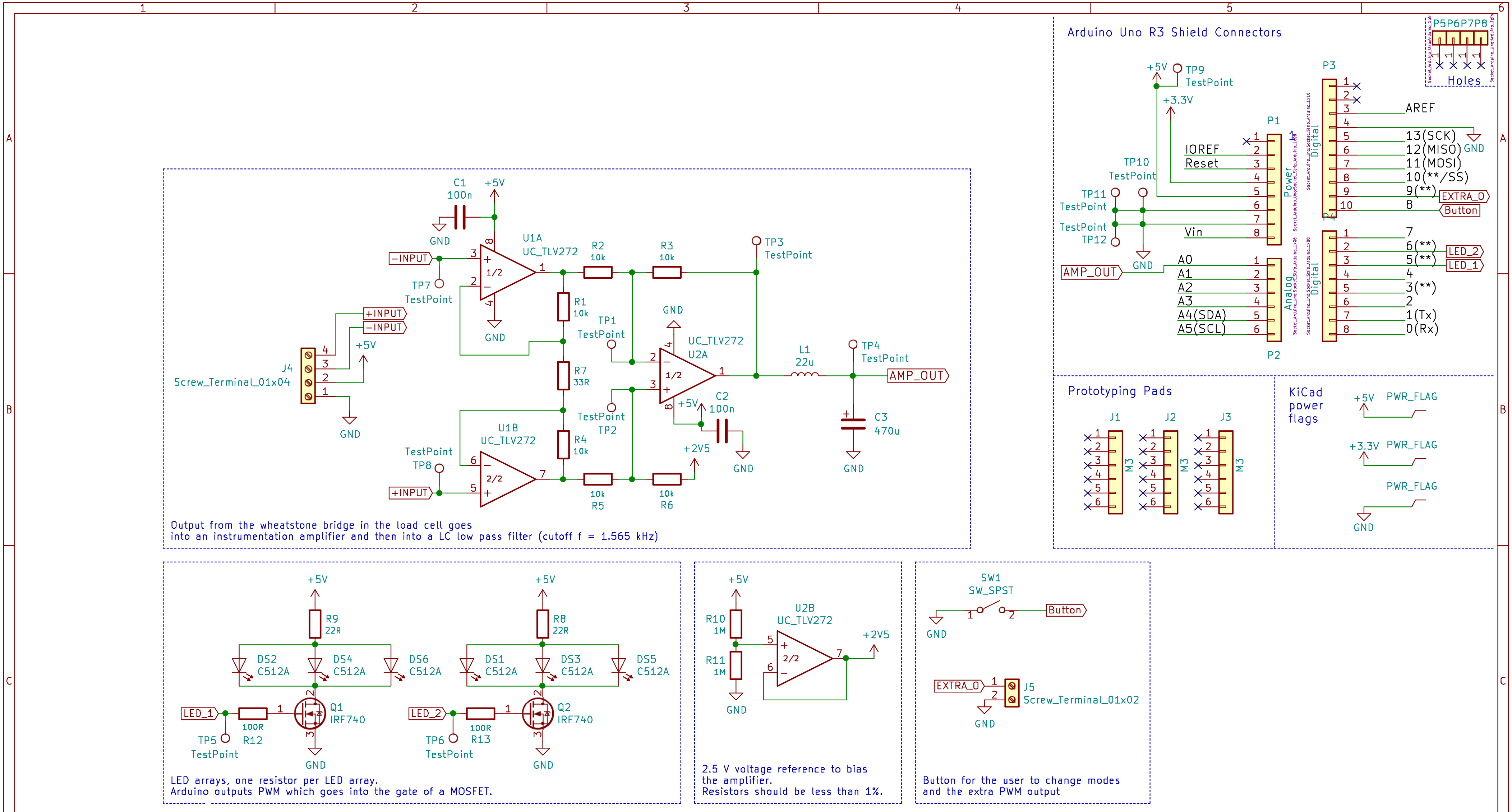
Listing 11: Voltage ADC header code.

```
/**
 * @file main.ino
 * @author Alex Stiles
 * @details
 * @brief Entry point. Contains high level logic and polling loop.
 * @version 0.1 alpha
 * @date 2021-08-13
 *
 * @copyright Copyright (c) 2021
 *
 * The entry point for compilation when compiling on Arduino.
 * Contains the setup() and loop() functions required for the Arduino.
 * C headers included are appropriately linked.
 *
 */

#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>

#include "voltage.h"

#include "LED_Group_Collection.h"
#include "LED_Group.h"
#include "Load_Cell.h"
#include "button.h"

#define LOAD_CELL_ANALOG_PIN 0

#define LEFT_GROUP_PIN 5
#define RIGHT_GROUP_PIN 6

#define BUTTON_PIN 8

LED_Group_Collection_t led_group_collection;
Load_Cell_t load_cell;
Button_t button;

/* Button */

void callbackFunction() {
    led_group_collection.mode = (Light_Mode_t) ((led_group_collection.mode + 1) %
        NUM_MODES);
    led_group_collection.mode_changed = true;
}

/* Load Cell */

float read_load_cell_voltage() {
    return (analogRead(LOAD_CELL_ANALOG_PIN) / (float) ARDUINO_PWM_READ_RANGE) *
        ARDUINO_MAX_VOLTAGE;
}

/* LED Groups */

void update_led_group_pwm_signals() {
    set_led_group_brightnesses(&led_group_collection, &load_cell);

    Serial.println("Left: " + String(get_group_brightness(get_led_group(&
        led_group_collection, LEFT))));
    Serial.println("Right: " + String(get_group_brightness(get_led_group(&
        led_group_collection, RIGHT))));
```

```

    analogWrite(LEFT_GROUP_PIN, get_group_brightness(get_led_group(&led_group_collection, LEFT)));
    analogWrite(RIGHT_GROUP_PIN, get_group_brightness(get_led_group(&led_group_collection, RIGHT)));
}

void setup() {
    Serial.begin(9600);

    led_group_collection = init_led_group_collection(LEFT_GROUP_PIN, RIGHT_GROUP_PIN);
    load_cell = init_load_cell(LOAD_CELL_ANALOG_PIN);
    button = init_button(BUTTON_PIN, &callbackFunction);

    pinMode(LOAD_CELL_ANALOG_PIN, INPUT);
    pinMode(BUTTON_PIN, INPUT_PULLUP);

    pinMode(LEFT_GROUP_PIN, OUTPUT);
    pinMode(RIGHT_GROUP_PIN, OUTPUT);
}

void loop() {
    long unsigned current_time = millis();
    bool button_state = digitalRead(BUTTON_PIN);

    debounce_state(&button, button_state, current_time);

    if (poll_sensor(&load_cell, read_load_cell_voltage()) || led_group_collection.mode_changed) {
        update_led_group_pwm_signals();

        Serial.println("Load Cell Voltage: " + String(load_cell.voltage));
        Serial.println("Load Cell Strain: " + String(load_cell.strain));
        Serial.println("Load Cell Angle: " + String(load_cell.angle));
    }

    delay(20);
}
```

Listing 12: Main code.

C Schematic



(75953196) & (18120890)

Edward Manson & Alex Stiles

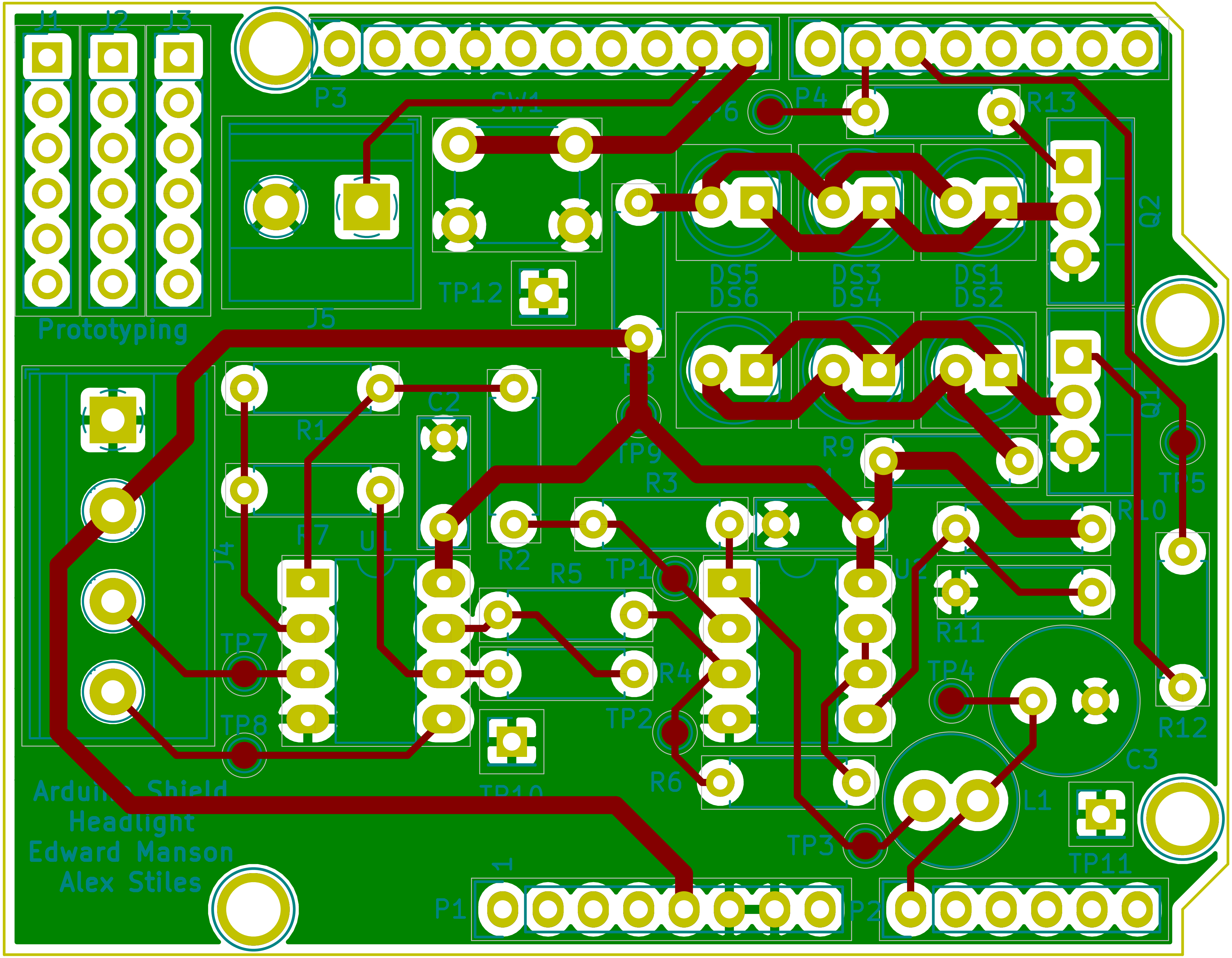
Sheet: /

File: ENEL200ShieldGroup10.sch

Title: Arduino Shield Headlight

Size: A4	Date: 2021-08-17	**Rev: A**
KiCad E.D.A. kicad 5.1.10-1.fc34		Id: 1/1

D Printed Circuit Board


J1
J2
J3
P3
SW1
TP6
P4
R13
Q2
DS5
DS6
DS3
DS4
DS1
DS2
TP12
J5
Prototyping
R8
R1
C2
TP9
R9
Q1
TP5
R3
R10
R7
U1
R2
R5
TP1
U2
J4
R11
TP7
R4
TP4
TP8
TP2
R12
C3
R6
L1
TP3
TP11
Shield
Headlight
Edward Manson
Alex Stiles
1
P1
P2