Rise And Shine

By Kevin Brown

Description of Product:

After moving from the Bay Area to San Diego, I no longer had direct sunlight glaring through my bedroom window every morning. Being a heavy sleeper, I have found that I am much more likely to wake up refreshed when awoken by light rather than sound. There are several products on the market that can help solve this issue through the automation of one's shades, however, they are quite expensive and typically work off of a physical controller or a timer. With this in mind, I wanted to automate my shades based on intensity of light outside. In addition to saving money, I thought it would be a good opportunity to incorporate several concepts I have been learning about in ME102B. I therefore designed my system to mount on the wall and actuate the shade to a desired position depending on how light it is outside. In addition, the system continuously monitors current draw to ensure that if the shade is stuck, the system will shut down.

Figure 1: Bedroom window without (left) and with (right) automation device. The device is adhered to the wall and the motor is connected to a 5V power supply.

Electromechanical Details:

Interfacing with existing Shade: The motor is mounted to the wall via a mount and connects directly to a roller shade clutch (Gear for ball chains) utilizing set screws. This clutch drives the ball chain that is wrapped around it. The clutch was sized based on required torque to raise and lower the shade along with space constraints. The torque required to actuate the shade was determined using a luggage scale and the radius of the shade tube. In addition, the motor was sized based on the aforementioned requirements along with available power supplies. The motor and motor clutch assembly can be seen in figure 2 below.

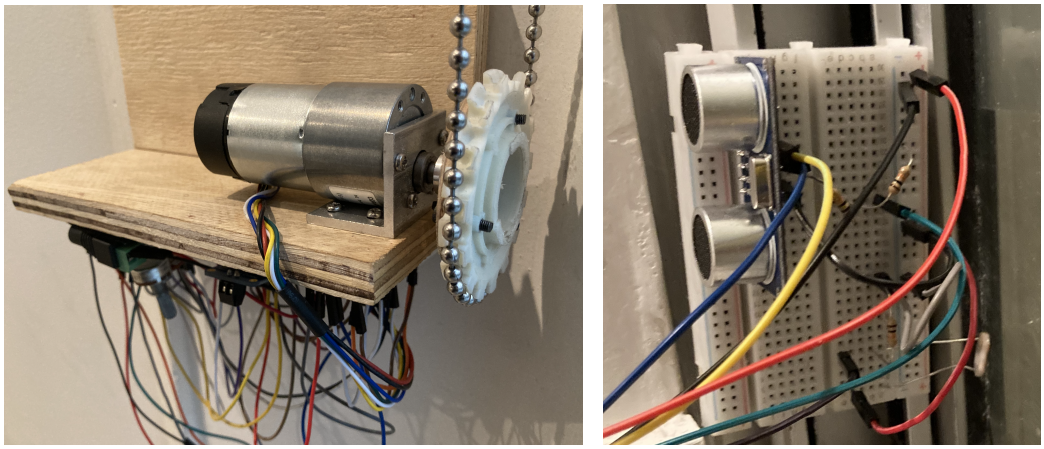

Figure 2: DC Motor to ball chain connection via roller shade clutch (left). Window sensor sub-assembly (right)

Motor Mount and Window Sensors: The motor mount, allows for the attachment of the motor to the wall along with space for the circuitry underneath. The cables for the window sensing unit, the USB cable and the power cable are all accessible from the bottom of the mount. The mount was placed so that the ball chain was pre-loaded and absent of slack to ensure the ball chain would not jump across any of the teeth. The window sensing unit is located away from the main mount and directly attached to the wall near the bottom of the window. The window sensing unit can be seen in figure 2 above.

Circuit:

The largest challenge of the project was setting up the interface between several components and ensuring the coded logic delivered the desired results when the aggregate system was built. There were (6) main components: (1) Brushed DC Motor, (2) Motor Driver, (3) Current Sensor, (4) Photo Resistor, (5) Ultrasonic sensor and (6) Potentiometer.

1. Brushed DC Motor: I purchased a CQRobot 270:1 Metal Gearmotor in-order to generate the necessary torque with the power supply I had available (5V, 2A). The DC motor came with a 64 count per revolution rotary encoder and all necessary wiring. The encoder was also used to track linear distance traveled and use this as an upper limit value for when the shade was in the retracting state.
2. DRV8833 Motor Driver: The DRV883 motor driver was provided in our lab kit. The driver allowed for multi-directional rotation of the DC motor along with dynamic breaking capabilities.
3. INA219 Breakout Board: I had an old INA219 from ME100 that was utilized . This component allowed for the monitoring of current draw to the motor driver. With this information a safety event was incorporated that would shut the motor off in the event

of rapidly increasing torque (in the case of a jam). This then required a manual reset of the ESP32 to go back to the non-emergency off state.

4. Photo Resistor: I purchased Photo Cells from Fry's electronics in-order to measure the light intensity outside. The photo cells act as the sensing device that commands the motors direction.
5. Ultrasonic Sensor: As with the INA219, I had an old HC-SR04 Ultrasonic Sonar Distance Sensor from ME100 that I utilized for my extension limit. This sensor was utilized to sense whether the shade at extended to the desired point and then shut the motor off.
6. Potentiometer: The Potentiometer was also part of our lab kit provided for ME102B. The potentiometer was utilized to set the DC motor speed and ensure that shade actuation could be achieved in a reasonable time.

Below a full circuit diagram can be seen. It should be noted that there are (3) resistors, each of which has the value denoted in the diagram. My 5V, 2A power supply is routed into the Vin pin on the INA219 and out to the DRV8822 motor driver, while the microcontroller is powered by a USB attached to my computer. Prior to energizing the circuit, a thorough review of the circuit was conducted to ensure the ESP32 was not in anyway connected to the 5V power supply.

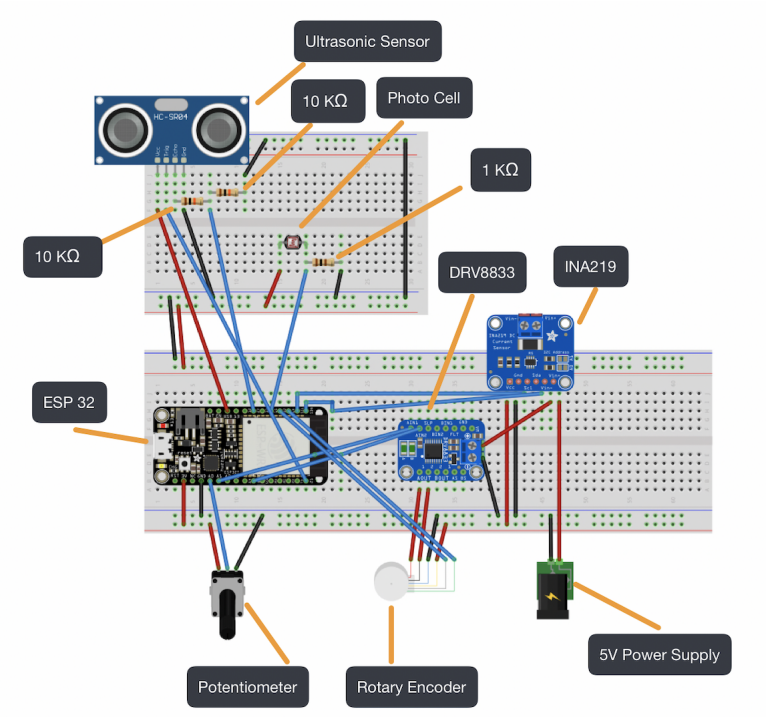

Figure 3: Rise And Shine Circuit Diagram

Finite State Machine

The system has several states it can be in depending on what events occur. These states and events can be seen in the finite state diagram below. From the FSD, the photocell, Ultrasonic sensor and current sensor required some calibration in-order to determine threshold values. The photocell threshold was acquired empirically by measuring values at several times of day and then determining when I wanted the shade to actuate, while the current and ultrasonic senors were matched against a DMM and ruler. In addition to the FSD, a validation plot can be seen below. It should be noted that the emergency off state was set to 1.1 Amps and the retract limit to 60 inches, these are not plotted below but were validated separately.

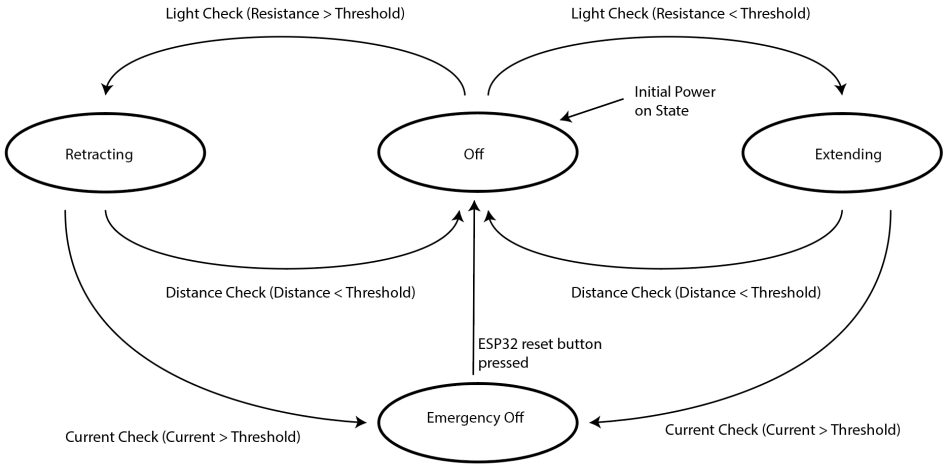

Figure 4: Finite State Diagram

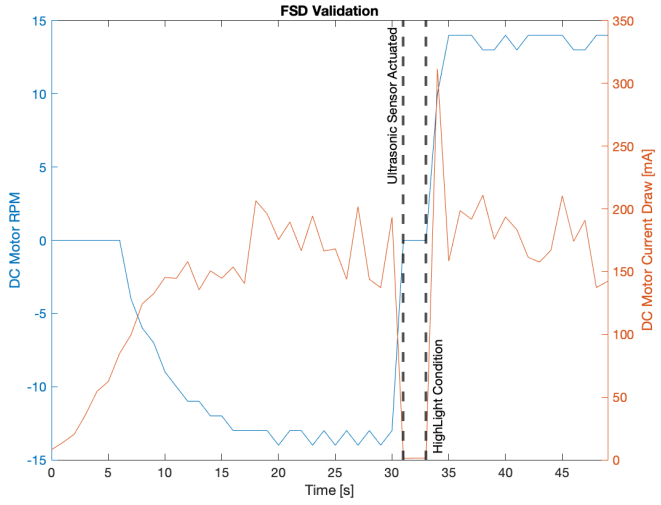

Figure 5: Motor RPM and Current draw over time from low light to high light conditions

Appendix I: Arduino Code

```
#include <Wire.h>
#include <ESP32Encoder.h>
#include <Adafruit_INA219.h>
ESP32Encoder encoder;
Adafruit_INA219 ina219;

// Constants and Variables

// Inputs
//#define interruptPin 21
#define Pot_Pin A0
#define Photo A10
//#define buzzer A12
#define trigPin 21
#define echoPin A8 // Pin 15

// Outputs
#define MPin_0 A1
#define MPin_1 A5

// PWM Specs
#define Freq 5000
#define Resolution 8
#define Led_Channel0 0
#define Led_Channel1 1

// Buzzer PWM Specs
#define Freqbuzz 1000
#define Resbuzz 8
#define Led_Channel2 2

// Feedback contral parameters
#define Kp 1.2
#define Ki 0.012
#define Feedback_Period 10000 // 10ms

// Thresholds
#define Debounce 200

// State Machine Variables
```

```
41 boolean state = 0;
42 volatile boolean interruptEvent = false;
43 bool printToggle = true;
44 bool emergency = false;
45
46 // Input Variables
47 int Potval = 0;
48 int duty = 0;
49 int tstep = 0; // t(k) variable
50 int photoValue = 0;
51
52 // Timer variables
53 volatile int buttonTimer = 0;
54 volatile int buttonTimer_prev = 0;
55
56 // PWM Duty Cycle Variables
57 int dutyCycle_state1 = 0;
58
59 // Encoder Variables
60 volatile int velGoal;
61 volatile int rpm;
62 volatile int count;
63 volatile int error;
64 volatile int errorSum;
65 volatile int Icont;
66 volatile int Pcont;
67
68 // INA219 Variables
69 float shuntvoltage = 0;
70 float busvoltage = 0;
71 float current_mA = 0;
72 float loadvoltage = 0;
73 float power_mW = 0;
74
75 // Ultrasonic Sensor Variables
76 float duration = 0;
77 int distlow = 0;
78 int disthigh = 0;
79 int dist = 0;
80
81 //Initializing timer with pointer
82 hw_timer_t * timer = NULL;
```

```
83 volatile SemaphoreHandle_t timerSemaphore;
84 portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED;
85 volatile uint32_t lastIsrAt = 0;
86
87 // Encoder Interrupt
88 void IRAM_ATTR encodercount()
89 {
90   portENTER_CRITICAL_ISR(&timerMux);
91   count = encoder.getCount();
92   printToggle = true;
93   encoder.clearCount();
94   error = velGoal - rpm;
95
96   //  PI Control Scheme
97   errorSum += error;
98   Pcont = Kp * error;
99   Icont = Kp * Ki * errorSum;
100  duty = Pcont + Icont;
101
102  // Direction
103  if ((photoValue < 500) && (distlow > 10))
104  {
105      dutyCycle_state1 = duty;
106      ledcWrite(Led_Channel0, dutyCycle_state1);
107      ledcWrite(Led_Channel1, 0);
108  }
109  else if ((photoValue < 500) && (distlow < 10))
110  {
111      ledcWrite(Led_Channel0, 0);
112      ledcWrite(Led_Channel1, 0);
113      disthigh = 0;
114  }
115
116  if ((photoValue > 500) && (disthigh < 60))
117  {
118      dutyCycle_state1 = duty;
119      ledcWrite(Led_Channel0, 0);
120      ledcWrite(Led_Channel1, dutyCycle_state1);
121  }
122  else if ((photoValue > 500) && (disthigh > 60))
123  {
124      ledcWrite(Led_Channel0, 0);
```

```
125       ledcWrite(Led_Channel1, 0);
126   }
127
128   //  // Anit Wind-up
129   //  if (duty > 255)
130   //  {
131   //    duty = 255;
132   //    dutyCycle_state1 = duty;
133   //    ledcWrite(Led_Channel0, dutyCycle_state1);
134   //    ledcWrite(Led_Channel1, 0);
135   //  }
136   //  else if (duty < 70)
137   //  {
138   //    duty = 70;
139   //    dutyCycle_state1 = duty;
140   //    ledcWrite(Led_Channel0, dutyCycle_state1);
141   //    ledcWrite(Led_Channel1, 0);
142   //  }
143
144   portEXIT_CRITICAL_ISR(&timerMux);
145   xSemaphoreGiveFromISR(timerSemaphore, NULL);
146 }
147
148 // Initiallization %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
149 void setup()
150 {
151
152   // Pin Setups
153   //  pinMode(interruptPin, INPUT_PULLUP);
154   pinMode(Photo, INPUT);
155   pinMode(trigPin, OUTPUT);
156   pinMode(echoPin, INPUT);
157   pinMode(Pot_Pin, INPUT);
158   pinMode(MPin_0, OUTPUT);
159   pinMode(MPin_1, OUTPUT);
160
161   // Interrupt Setup
162   //Button interrupt to turn motor on and off
163   //  attachInterrupt(digitalPinToInterrupt(interruptPin), button,
        RISING);
164
165   // Configure PWM
```

```
166   ledcSetup(Led_Channel0, Freq, Resolution);
167   ledcSetup(Led_Channel1, Freq, Resolution);
168   // Attach the channel to the GPIO to be controlled
169   ledcAttachPin(MPin_0, Led_Channel0);
170   ledcAttachPin(MPin_1, Led_Channel1);
171   // Initially write to PWM a duty cycle of 0
172   ledcWrite(Led_Channel0, 0);
173   ledcWrite(Led_Channel1, 0);
174
175   // Configure Buzzer PWM
176   //  ledcSetup(Led_Channel2, Freqbuzz, Resbuzz);
177   //  ledcAttachPin(buzzer, Led_Channel2);
178
179   // Configure Timer interrupt
180   // Timer setup for encodercount
181   timerSemaphore = xSemaphoreCreateBinary();
182   timer = timerBegin(0, 80, true); //timer 1, prescaler of 80,
        counts up
183   timerAttachInterrupt(timer, &encodercount, true);
184   timerAlarmWrite(timer, Feedback_Period, true); //interrupt every
        0.01 second (0.5 Hz)
185
186   // Encoder Setup
187   //ESP32Encoder::useInternalWeakPullResistors=DOWN;
188   // Enable the weak pull up resistors
189   ESP32Encoder::useInternalWeakPullResistors = UP;
190   // Attache pins for use as encoder pins
191   encoder.attachFullQuad(19, 18);
192   // set starting count value after attaching
193   encoder.setCount(0);
194   // clear the encoder's raw count and set the tracked count to
        zero
195   encoder.clearCount();
196   // get the endcoder count
197   encoder.getCount();
198
199   // INA219 Setup
200   uint32_t currentFrequency;
201
202   if (! ina219.begin()) {
203     Serial.println("Failed to find INA219 chip");
204     while (1) {
```

```
205       delay(10);
206     }
207   }
208   //To use a slightly lower 32V, 1A range (higher precision on amps
        ):
209   ina219.setCalibration_32V_1A();
210
211   Serial.begin(115200);
212 }
213
214 // Loop %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
215 void loop()
216 {
217
218   // Photo Resistor Readings
219   photoValue = analogRead(Photo);
220   digitalWrite(trigPin, LOW);
221   delay(10);
222   digitalWrite(trigPin, HIGH);
223   delay(10);
224   digitalWrite(trigPin, LOW);
225   duration = pulseIn(echoPin, HIGH);
226   distlow = (duration * .0343) / 2;
227
228
229   if (current_mA > 1100)
230   {
231     service2();
232   }
233   if ((current_mA < 1100))
234   {
235     service1();
236
237     switch (printToggle)
238     {
239       case true:
240         shuntvoltage = ina219.getShuntVoltage_mV();
241         busvoltage = ina219.getBusVoltage_V();
242         current_mA = ina219.getCurrent_mA();
243         power_mW = ina219.getPower_mW();
244         loadvoltage = busvoltage + (shuntvoltage / 1000);
245         tstep = (tstep + 1); // t(k) counter
```

```
246         rpm = 0.0003636 * abs(count) * 6000 / (2 * PI); // Actual
               RPM from encoder
247         dist = 0.11*abs(count);
248         disthigh = disthigh+dist;
249         // Serial Prints when motor is on
250         Serial.print("Light:␣␣"); Serial.println(photoValue);
251         //Serial.print("Time:   "); Serial.println(tstep); // Time
               step
252         Serial.print("RPM:␣␣␣"); Serial.println(rpm); // Encoder
               RPM
253         Serial.print("Dist:␣"); Serial.println(dist);
254         Serial.print("Distance:␣"); Serial.println(distlow);
255         Serial.print("Distance␣High:␣"); Serial.println(disthigh);
256         Serial.print("VelGoal:␣␣"); Serial.println(velGoal); //
               Goal Velocity
257         //Serial.print("Bus Voltage:   "); Serial.print(busvoltage)
               ; Serial.println(" V");
258         //Serial.print("Shunt Voltage: "); Serial.print(
               shuntvoltage); Serial.println(" mV");
259         //Serial.print("Load Voltage:  "); Serial.print(loadvoltage
               ); Serial.println(" V");
260         Serial.print("Current:␣␣␣␣␣␣␣"); Serial.print(current_mA);
               Serial.println("␣mA");
261         Serial.print('\n');
262         delay(1000);
263         printToggle = false;
264         break;
265
266       default:
267         break;
268     }
269     // Pot readings
270     Potval = analogRead(Pot_Pin);
271     // Maps analog reading to min ---> max RPM
272     velGoal = map(Potval, 0, 4095, 0, 15);
273   }
274
275 }
276
277
278 // Service Routines %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
279 void service0()
```

```
280 {
281   // Turn off motor
282   timerAlarmDisable(timer);
283   ledcWrite(Led_Channel0, 0);
284   ledcWrite(Led_Channel1, 0);
285 }
286
287 void service1()
288 {
289   // Turn on motor
290   timerAlarmEnable(timer); //starts alarm
291 }
292
293 void service2()
294 {
295   timerAlarmDisable(timer);
296   ledcWrite(Led_Channel0, 0);
297   ledcWrite(Led_Channel1, 0);
298   emergency = true;
299 }
```