FORMULA 1 SAFETY CAR WARNING LIGHT SYSTEM

Replicated using an Arduino board

Dean Lonergan

Automotive Software Concepts

Feature list:

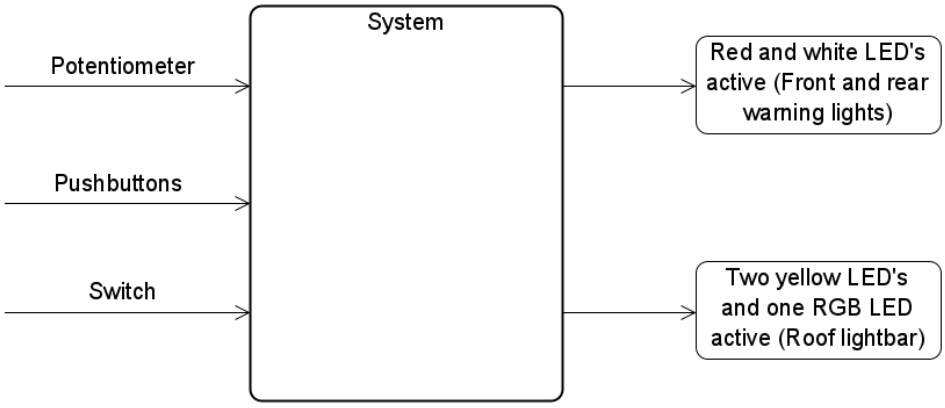
The user will be able to:

1. Control the front and rear warning lights.
2. Enable and control the roof-mounted lightbar.
3. Change the speed at which the front and rear warning lights flash.
4. Change between warning light colours of the roof-mounted lightbar.

Non-Functional Requirements:

- This system must be battery powered and reliable.
- It must be installable in a suitable vehicle; is not cumbersome.
- Operating the system should be intuitive.
- It must always remain alert for user inputs, capable of changing outputs at a moment's notice without delay.

Context diagram:



Name	Direction	Type	Description
Potentiometer	Input	Analogue	Analogue user input to enable and control the front and rear warning lights.
Pushbuttons	Input	Digital	Digital user input to control the colour of the lightbar warning light.
Switch	Input	Digital	Digital user input to enable/disable the lightbar.
Red and white LED's	Output	Analogue (PWM)	Flashing speed can be altered based on user input.
Yellow and RGB LED's	Output	Analogue (PWM)	Can display a specific warning colour based on user input.

List of states:

- **Init** – The system is initialized.

- **Front and rear lights off** - The system waits for user input.
- **Front and rear lights on** - The system enables the front and rear warning lights which flash at a speed controlled by the user.

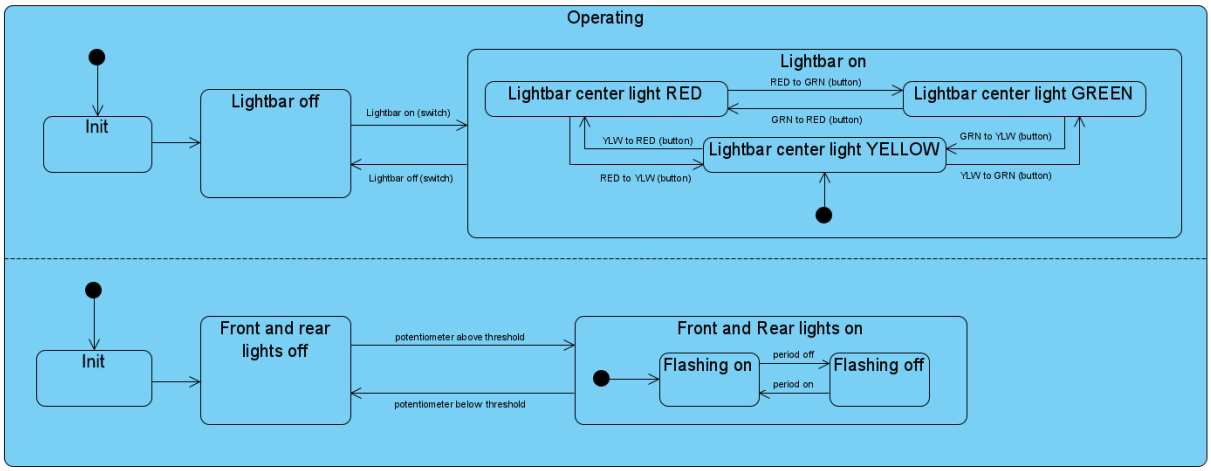
- **Lightbar off** – The system waits for user input.
- **Lightbar on** – The system enables the roof lightbar and the centre light. The colour of the centre light is controlled by the user.
    - **Lightbar centre light yellow** - The default state of the centre LED when the lightbar is activated.
    - **Lightbar centre light green** – Can be switched to by the user.
    - **Lightbar centre light red** – Can be switched to by the user.

List of events:

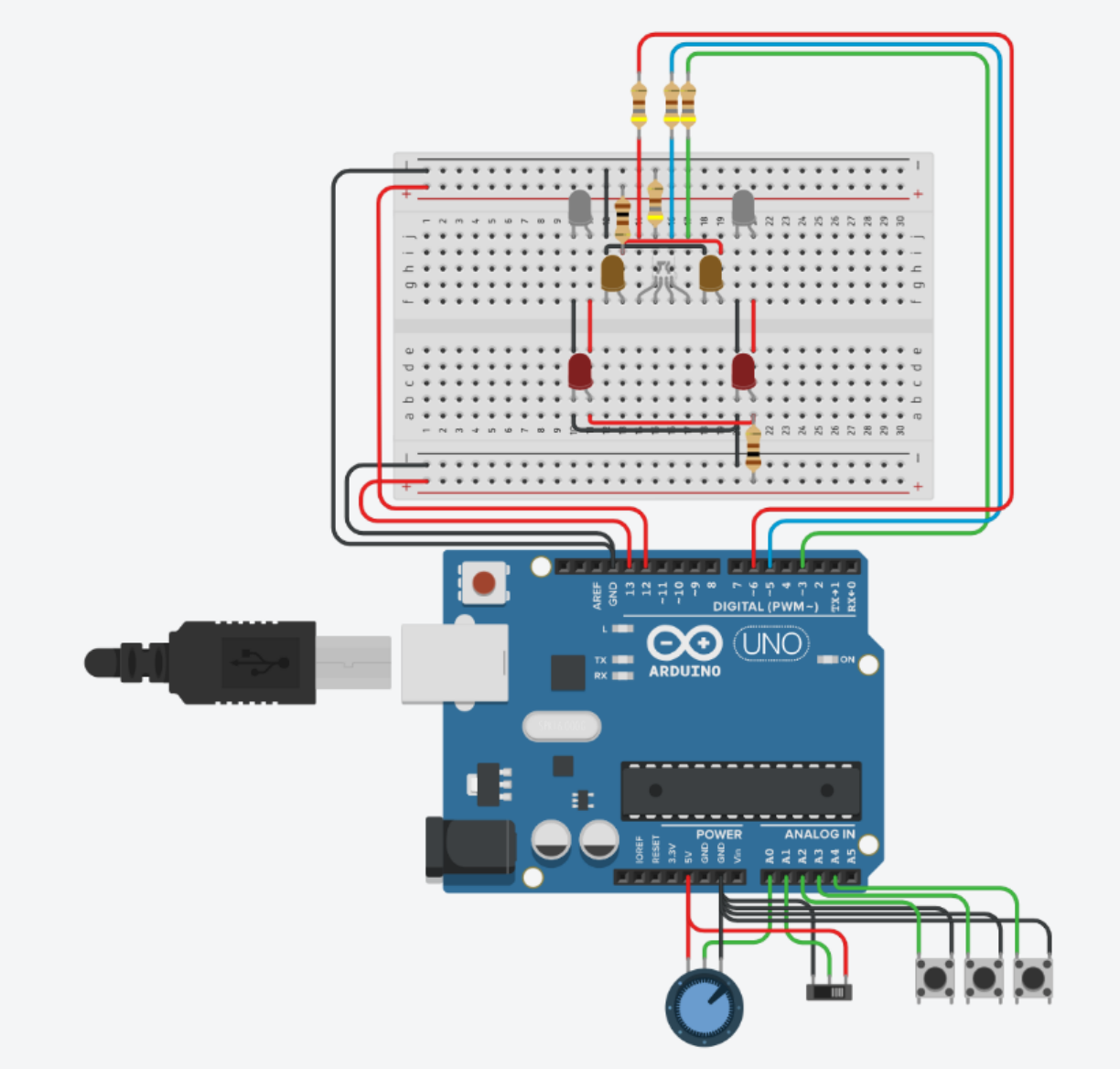
- **potentiometer above threshold** – The user has raised the potentiometer above a determined threshold, enabling the front and rear lights.
- **potentiometer below threshold** – The user has lowered the potentiometer below a determined threshold, disabling the front and rear lights.

- **switch on the lightbar** – The user enables the roof mounted lightbar using a switch.
    - **switch to the yellow centre light** – The default state of the centre LED when the lightbar is activated.
    - **switch to the green centre light** – The user presses a button to swap to a different colour.
    - **switch to the red centre light** – The user presses a button to swap to a different colour.
- **switch off the lightbar** – The user disables the roof mounted lightbar using a switch.

System statechart:



Completed system:



System code:

```
#define ST_INIT           0

#define ST_LIGHTBAR_OFF   1

#define ST_LIGHTBAR_ON_Y  2

#define ST_LIGHTBAR_ON_G  3

#define ST_LIGHTBAR_ON_R  4

#define ST_FR_LIGHTS_OFF  1

#define ST_FR_LIGHTS_ON   2

#define FR_LIGHTS_PTNM_IN A0

#define LGTBR_SW_IN       A1

#define LGTBR_RED_BN_IN   A2

#define LGTBR_GRN_BN_IN   A3

#define LGTBR_YLW_BN_IN   A4

#define FR_LIGHTS_OUT     13

#define LGTBR_OUT         12

#define RGB_RED_OUT       6

#define RGB_BLU_OUT       5

#define RGB_GRN_OUT       3

#define ON                1

#define OFF               0

#define MAX               255

#define MIN               0

unsigned long currentTime;

unsigned long previousTime;

int currentState_FR_LIGHTS;

int currentState_LIGHTBAR;

int sensorValue;

int flashInterval;

int flashState;

int redButton;

int greenButton;

int yellowButton;

int lightbarSwitch;
```

```
void setup(){
  currentState_FR_LIGHTS = ST_INIT;
  currentState_LIGHTBAR = ST_INIT;
  pinMode(FR_LIGHTS_PTNM_IN, INPUT);
  pinMode(LGTBR_SW_IN, INPUT);
  pinMode(LGTBR_RED_BN_IN, INPUT_PULLUP);
  pinMode(LGTBR_GRN_BN_IN, INPUT_PULLUP);
  pinMode(LGTBR_YLW_BN_IN, INPUT_PULLUP);
  pinMode(FR_LIGHTS_OUT, OUTPUT);
  pinMode(LGTBR_OUT, OUTPUT);
  pinMode(RGB_RED_OUT, OUTPUT);
  pinMode(RGB_BLU_OUT, OUTPUT);
  pinMode(RGB_GRN_OUT, OUTPUT);
  currentState_FR_LIGHTS = ST_FR_LIGHTS_OFF;
  currentState_LIGHTBAR = ST_LIGHTBAR_OFF;
}

void loop(){
  currentTime = millis();
  sensorValue = analogRead(FR_LIGHTS_PTNM_IN);
  lightbarSwitch = digitalRead(LGTBR_SW_IN);
  redButton = !digitalRead(LGTBR_RED_BN_IN);
  greenButton = !digitalRead(LGTBR_GRN_BN_IN);
  yellowButton = !digitalRead(LGTBR_YLW_BN_IN);

  //Front-Rear Lights - State 1//
  if (currentState_FR_LIGHTS == ST_FR_LIGHTS_OFF) {
    if (sensorValue > 100) {
      currentState_FR_LIGHTS = ST_FR_LIGHTS_ON;
    } else {
      digitalWrite(FR_LIGHTS_OUT, LOW);
    }
  }
```

```
//Front-Rear Lights - State 2//
if (currentState_FR_LIGHTS == ST_FR_LIGHTS_ON) {
  if (sensorValue < 100) {
    currentState_FR_LIGHTS = ST_FR_LIGHTS_OFF;
  } else {
    if ((currentTime - previousTime) >= flashInterval) {
      previousTime = currentTime;
      if (flashState == OFF) {
        flashState = ON;
      } else {
        flashState = OFF;
      }
    }
    digitalWrite(FR_LIGHTS_OUT, flashState);
  }
  if (sensorValue > 100 && sensorValue < 260) {
    flashInterval = 300;
  } else if (sensorValue > 260 && sensorValue < 430) {
    flashInterval = 250;
  } else if (sensorValue > 430 && sensorValue < 600) {
    flashInterval = 200;
  } else if (sensorValue > 600 && sensorValue < 780) {
    flashInterval = 150;
  } else if (sensorValue > 780 && sensorValue < 1000) {
    flashInterval = 100;
  } else if (sensorValue > 1000){
    flashInterval = 0;
  }
}
```

```
//Lightbar - State 1//
if (currentState_LIGHTBAR == ST_LIGHTBAR_OFF) {
  if (lightbarSwitch) {
    currentState_LIGHTBAR = ST_LIGHTBAR_ON_Y;
  } else {
    digitalWrite(LGTBR_OUT, OFF);
    analogWrite(RGB_RED_OUT, MIN);
    analogWrite(RGB_BLU_OUT, MIN);
    analogWrite(RGB_GRN_OUT, MIN);
  }
}

//Lightbar - State 2//
if (currentState_LIGHTBAR == ST_LIGHTBAR_ON_Y) {
  digitalWrite(LGTBR_OUT, ON);
  analogWrite(RGB_RED_OUT, MAX);
  analogWrite(RGB_BLU_OUT, MIN);
  analogWrite(RGB_GRN_OUT, MAX);
  if (redButton) {
    currentState_LIGHTBAR = ST_LIGHTBAR_ON_R;
  }
  if (greenButton) {
    currentState_LIGHTBAR = ST_LIGHTBAR_ON_G;
  }
  if (!lightbarSwitch) {
    currentState_LIGHTBAR = ST_LIGHTBAR_OFF;
  }
}
```

```
  //Lightbar - State 3//
  if (currentState_LIGHTBAR == ST_LIGHTBAR_ON_G) {
    analogWrite(RGB_RED_OUT, MIN);
    analogWrite(RGB_BLU_OUT, MIN);
    analogWrite(RGB_GRN_OUT, MAX);
    if (redButton) {
      currentState_LIGHTBAR = ST_LIGHTBAR_ON_R;
    }
    if (yellowButton) {
      currentState_LIGHTBAR = ST_LIGHTBAR_ON_Y;
    }
    if (!lightbarSwitch) {
      currentState_LIGHTBAR = ST_LIGHTBAR_OFF;
    }
  }

  //Lightbar - State 4//
  if (currentState_LIGHTBAR == ST_LIGHTBAR_ON_R) {
    analogWrite(RGB_RED_OUT, MAX);
    analogWrite(RGB_BLU_OUT, MIN);
    analogWrite(RGB_GRN_OUT, MIN);
    if (greenButton) {
      currentState_LIGHTBAR = ST_LIGHTBAR_ON_G;
    }
    if (yellowButton) {
      currentState_LIGHTBAR = ST_LIGHTBAR_ON_Y;
    }
    if (!lightbarSwitch) {
      currentState_LIGHTBAR = ST_LIGHTBAR_OFF;
    }
  }
}
```

Code description:

The code for this system has been designed to be robust, non-blocking, and human-readable. As such, the code contains many definitions. These definitions improve readability and serve to make the code more adaptable. For example, if you are required to change a pin number, the change must only be made to the definition, rather than at every point in the code where the pin is referenced. States were also defined and assigned numbers, again to aid in readability. The resulting code can be readily adapted and easily understood.

The functionality was to be as straightforward as possible. An overly complex system was not desired as intuitive operation and reliability were paramount. The system is initialized in the setup method, the Arduino board is told which pins are inputs and which are outputs, and the subsystems are set to their default off states, awaiting user input. The loop method then begins running in perpetuity. Within the loop method, some further definitions are made. These track the time passed and the states of various inputs. These are defined here as they will change during operation and will help the readability of the code.

When both subsystems are in the off state, the system can only detect inputs from two sources, the potentiometer, and the slide switch. When an input from one of these is detected, for example, the slide switch is switched on, the lightbar subsystem enters the ST_LIGHTBAR_ON_Y state. When in this state, the code continues to loop and detect changes to the slide switch and the potentiometer but now also detect inputs from two of the three pushbuttons. The system will not detect input from the yellow pushbutton, as in this state the RGB LED is already yellow. If the red pushbutton is pressed while in this state, the system enters an equivalent ST_LIGHTBAR_ON_R state.

If the user raises the potentiometer above a certain threshold, the front and rear lights subsystem will enter the ST_FR_LIGHTS_ON state. When in this state, the code continues to loop but will also track the time passed compared to the current time, flashing the front and rear lights at the appropriate interval. The interval is changed depending on the current position of the potentiometer. This is achieved in a non-blocking manner, so as this takes place the code can continue to detect other inputs. Should the user raise the potentiometer to its maximum setting, the lights stop flashing and stay on statically. When the potentiometer is lowered again to its minimum setting, the subsystem returns to the ST_FR_LIGHTS_OFF state.

Testing Procedures:

Due to the nature of this system, its design had to be robust but simple. It needed to operate reliably and respond to user input instantly. As such, the system was designed using states. Each state could only be entered by specific user inputs and did not contain any blocking code. This was to mitigate errors, be they within the system or through improper user input. Testing required each state to be checked, first while operating independently and again while operating simultaneously.

I added code that printed relevant information about each state to the console. Then, using pen and paper, I went through each state one after the other and checked that it operated as intended, responded correctly and reliably to relevant user inputs, and ignored incorrect or irrelevant user inputs. I checked each state of both subsystems as they operated independently, and again as they operated simultaneously.

I found that no code blocked or delayed a response to user input. I found that the system operates reliably and does not respond unexpectedly. Each subsystem operates entirely independently, with no input or output from one affecting the other. After this testing, I am confident that the system operates as intended.

Final Functionality:

The system first initializes; when initialized, each subsystem enters its respective off state, awaiting user input. If the user raises the potentiometer beyond a certain threshold, the front and rear light subsystem will enter the ST_FR_LIGHTS_ON state. In this state, the LEDs representing the front and rear lights of the safety car will flash at a speed determined by the potentiometer input, up to a maximum input point at which they will remain static. If the user switches the lightbar on, the lightbar subsystem will enter the ST_LIGHTBAR_ON_Y state.

When in the ST_LIGHTBAR_ON_Y state, the two orange LEDs representing the lightbar are solid while the centre RGB LED is yellow. While the lightbar is on, the user can press the pushbuttons to change the colour of the centre RGB LED from yellow to red or green. If the lightbar is switched off the lightbar state will be changed to ST_LIGHTBAR_OFF. If the potentiometer is set to below the threshold, the front and rear light state will change to off. These subsystems have been implemented to operate simultaneously and independently, with non-blocking code.