```
/*****************************************************************************************
   Slotcar Race Controller for PCLapCounter Software

   (C) Copyright 2016-2017 el.Dude - www.eldude.nl

   Arduino MEGA 2560 based slotcar race controller. Capture start/finish signals,
   controls the power relays as well as any signal LEDs and manages external buttons.

   See http://pclapcounter.be/arduino.html for the input/output protocol.
   Minimum PC Lap Counter version: 5.40

   Author: Gabriel Inäbnit
   Date  : 2016-10-14

   TODO:
   - Multi heat race proper false start and heat end detection
   - disable track call button when race is not active (or change button behaviour)
   - aborting start/restart is bogus

   Revision History
   __________ _______________________ _______________________________________________________
   2017-01-25 Gabriel Inäbnit      Light show pattern functionality
   2017-01-22 Gabriel Inäbnit      LEDs and Relay code refactored with classes
   2017-01-21 Gabriel Inäbnit      Lane detection blackout period added
   2017-01-17 Gabriel Inäbnit      Interrupt to Lane mapping also configured with array
   2017-01-16 Gabriel Inäbnit      Relays NC, r/g/y racer’s stand lights, lane mappings
   2016-10-31 Gabriel Inäbnit      Race Clock - Race Finished status (RC2) PCLC v5.40
   2016-10-28 Gabriel Inäbnit      Start/Finish lights on/off/blink depending race status
   2016-10-25 Gabriel Inäbnit      Removed false start init button - no longer needed
   2016-10-24 Gabriel Inäbnit      Fix false start GO command with HW false start enabled
   2016-10-22 Gabriel Inäbnit      HW false start enable/disable, penalty, reset
   2016-10-21 Gabriel Inäbnit      false start detection and penalty procedure
   2016-10-18 Gabriel Inäbnit      external buttons handling added
   2016-10-14 Gabriel Inäbnit      initial version
 *****************************************************************************************/

/*****************************************************************************************
   Do not use pins:
   Serial1: 18 & 19 - used for interrupts
   Serial2: 16 & 17
   Serial3: 14 & 15
   BuiltIn: 13 - try to avoid it
 *****************************************************************************************/

/*****************************************************************************************
   Global variables
 *****************************************************************************************/
const long serialSpeed = 19200; // 19200;
const long serial3Speed = 115200; // bluetooth
const unsigned long laneProtectionTime = 9000L;
const byte laneToInterrupMapping[] = { 18, 19, 20, 21,  3,  2 };
const byte laneToRelayMapping[]    = { 12, 28, 11,  9,  7,  5 };
const byte laneToGreenMapping[]    = { 44, 46, 38, 34, 39, 35 };
const byte laneToRedMapping[]      = { 41, 42, 40, 36, 32, 37 };
const char lapTime[][7] =
{
  "[SF01$",
  "[SF02$",
  "[SF03$",
  "[SF04$",
  "[SF05$",
  "[SF06$"
};

const unsigned long delayMillis[] =
{ // index
  0L, // 0
  1000L, // 1
  2000L, // 2
  3000L, // 3
  4000L, // 4
  5000L, // 5
  6000L, // 6
  7000L  // 7
};

/*****************************************************************************************
   Light Show
   patter positions:
   0              1   2    3    4    5    6    7   8   9   10  11  12
   _____________  __  ___  ___  ___  ___  ___  __  __  __  __  __  __
   Duration [ms], Go, SF5, SF4, SF3, SF2, SF1, R6, R5, R4, R3, R2, R1 (STOP & CAUTION n/c)

   values: 0 = off, 1 = on (red), 2 = green (only for R1..R6), 3 = yellow (only for R1..R6)
 *******************************************************************************************/
#define PATTERN_COLUMNS  13
const byte initPattern[][PATTERN_COLUMNS] = {
  {50, 1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 3},
  {50, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2},
  {50, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
  {50, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2},
  {50, 1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 3},
  {50, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2},
  {50, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
  {50, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  {50, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  {50, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  {50, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0},
  {50, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0},
  {50, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0},
  {50, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0},
  {50, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0},
  {50, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0},
  {50, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0},
  {50, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0},
  {50, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
  {50, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
  {50, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1},
  {50, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1},
  {50, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1},
  {50, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1},
  {50, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1},
  {50, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1},
  {50, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1},
  {50, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},
  {50, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
  {50, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
};

const byte showPattern[][PATTERN_COLUMNS] = {
  {66, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0},
  {66, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0},
  {66, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0},
  {66, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0},
  {66, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0},
  {66, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},
  {66, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1},
  {66, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1},
  {66, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0},
  {66, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0},
  {66, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0},
  {66, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0},
  {66, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0},
  {66, 1, 0, 1, 0, 1, 0, 2, 0, 0, 0, 0, 0},
  {66, 0, 1, 0, 1, 0, 1, 2, 2, 0, 0, 0, 0},
  {66, 1, 0, 1, 0, 1, 0, 0, 2, 2, 0, 0, 0},
  {66, 0, 1, 0, 1, 0, 1, 0, 0, 2, 2, 0, 0},
  {66, 1, 0, 1, 0, 1, 0, 0, 0, 0, 2, 2, 0},
  {66, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2},
  {66, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 2},
  {66, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 2},
  {66, 0, 0, 0, 0, 1, 0, 0, 0, 0, 2, 2, 0},
  {66, 0, 0, 0, 0, 0, 1, 0, 0, 2, 2, 0, 0},
  {66, 1, 0, 0, 0, 0, 0, 0, 2, 2, 0, 0, 0},
  {66, 0, 0, 0, 0, 0, 1, 2, 2, 0, 0, 0, 0},
  {66, 0, 0, 0, 0, 1, 0, 2, 0, 0, 0, 0, 0},
  {66, 0, 0, 0, 1, 0, 0, 3, 0, 0, 0, 0, 0},
  {66, 0, 0, 1, 0, 0, 0, 3, 3, 0, 0, 0, 0},
  {66, 0, 1, 0, 0, 0, 0, 0, 3, 3, 0, 0, 0},
  {66, 1, 0, 0, 0, 0, 0, 0, 0, 3, 3, 0, 0},
  {66, 0, 1, 0, 1, 0, 1, 0, 0, 0, 3, 3, 0},
  {66, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 3, 3},
  {66, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 3},
  {66, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 3, 3},
  {66, 0, 1, 0, 1, 0, 1, 0, 0, 0, 3, 3, 0},
    {66, 1, 0, 1, 0, 1, 0, 0, 0, 3, 3, 0, 0},
    {66, 0, 1, 0, 0, 0, 0, 0, 3, 3, 0, 0, 0},
    {66, 0, 0, 1, 0, 0, 0, 3, 3, 0, 0, 0, 0},
    {66, 0, 0, 0, 1, 0, 0, 3, 0, 0, 0, 0, 0},
    {66, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0},
    {66, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0},
    {66, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1},
    {66, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0},
    {66, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1},
    {66, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0},
    {66, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1},
    {66, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0},
    {66, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1},
    {66, 0, 1, 0, 1, 0, 1, 3, 0, 3, 0, 3, 0},
    {66, 1, 0, 1, 0, 1, 0, 0, 3, 0, 3, 0, 3},
    {66, 0, 1, 0, 1, 0, 1, 3, 0, 3, 0, 3, 0},
    {66, 1, 0, 1, 0, 1, 0, 0, 2, 0, 2, 0, 2},
    {66, 0, 1, 0, 1, 0, 1, 2, 0, 2, 0, 2, 0},
    {66, 1, 0, 1, 0, 1, 0, 0, 2, 0, 2, 0, 2},
    {66, 0, 1, 0, 1, 0, 1, 2, 0, 2, 0, 2, 0},
    {66, 1, 0, 1, 0, 1, 0, 0, 2, 0, 2, 0, 2},
    {66, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1},
    {66, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1},
    {66, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1},
    {66, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0},
    {66, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1},
    {66, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0},
    {66, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1},
    {66, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0},
    {66, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1},
    {66, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0},
    {66, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1},
    {255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
};

/*****************************************************************************************
   Arduono Button Press Messages
 *****************************************************************************************/
#define BUTTON_RACE_START          "[BT01]"
#define BUTTON_RACE_RESTART        "[BT02]"
#define BUTTON_RACE_PAUSE          "[BT03]"
#define BUTTON_RACE_NEXT           "[BT04]"
#define BUTTON_POWER_OFF           "[BT05]"
#define BUTTON_POWER_ON            "[BT06]"
#define BUTTON_END_OF_RACE         "[BT07]"
#define BUTTON_TOGGLE_POWER        "[BT08]"
#define BUTTON_TOGGLE_YELLOW_FLAG  "[BT09]"
#define BUTTON_STOP_AND_GO_LANE1   "[SG01]"
#define BUTTON_STOP_AND_GO_LANE2   "[SG02]"
#define BUTTON_STOP_AND_GO_LANE3   "[SG03]"
#define BUTTON_STOP_AND_GO_LANE4   "[SG04]"
#define BUTTON_STOP_AND_GO_LANE5   "[SG05]"
#define BUTTON_STOP_AND_GO_LANE6   "[SG06]"

/*****************************************************************************************
   Pin Naming
 *****************************************************************************************/
// lane to interrup pin mapping
#define LANE_1 laneToInterrupMapping[0]
#define LANE_2 laneToInterrupMapping[1]
#define LANE_3 laneToInterrupMapping[2]
#define LANE_4 laneToInterrupMapping[3]
#define LANE_5 laneToInterrupMapping[4]
#define LANE_6 laneToInterrupMapping[5]

#define LED_1 23
#define LED_2 25
#define LED_3 27
#define LED_4 29
#define LED_5 31

#define LED_DSR1 41
#define LED_DSG1 44
#define LED_DSR2 42
#define LED_DSG2 46
#define LED_DSR3 40
#define LED_DSG3 38
#define LED_DSR4 36
```

```
#define LED_DSG4 34
#define LED_DSR5 32
#define LED_DSG5 39
#define LED_DSR6 37
#define LED_DSG6 35

#define LED_STOP 22
#define LED_CAUTION 24
#define LED_GO 26

// PWR_x: x = lane
#define PWR_ALL 30
#define PWR_1   laneToRelayMapping[0] // 12
#define PWR_2   laneToRelayMapping[1] // 28
#define PWR_3   laneToRelayMapping[2] // 11
#define PWR_4   laneToRelayMapping[3] //  9
#define PWR_5   laneToRelayMapping[4] //  7
#define PWR_6   laneToRelayMapping[5] //  5

#define FSbit_0 10
#define FSbit_1 8
#define FSbit_2 6
#define FSbit_3 4

/*****************************************************************************************
   PC Lap Counter Messages
 *****************************************************************************************/
#define SL_1_ON  "SL011"
#define SL_1_OFF "SL010"
#define SL_2_ON  "SL021"
#define SL_2_OFF "SL020"
#define SL_3_ON  "SL031"
#define SL_3_OFF "SL030"
#define SL_4_ON  "SL041"
#define SL_4_OFF "SL040"
#define SL_5_ON  "SL051"
#define SL_5_OFF "SL050"

#define GO_ON       "SL061"
#define GO_OFF      "SL060"
#define STOP_ON     "SL071"
#define STOP_OFF    "SL070"
#define CAUTION_ON  "SL081"
#define CAUTION_OFF "SL080"

#define PWR_ON    "PW001"
#define PWR_OFF   "PW000"
#define PWR_1_ON  "PW011"
#define PWR_1_OFF "PW010"
#define PWR_2_ON  "PW021"
#define PWR_2_OFF "PW020"
#define PWR_3_ON  "PW031"
#define PWR_3_OFF "PW030"
#define PWR_4_ON  "PW041"
#define PWR_4_OFF "PW040"
#define PWR_5_ON  "PW051"
#define PWR_5_OFF "PW050"
#define PWR_6_ON  "PW061"
#define PWR_6_OFF "PW060"

/*****************************************************************************************
   Class Race
 *****************************************************************************************/
#define RACE_INIT '0'
#define RACE_STARTED '1'
#define RACE_FINISHED '2'
#define RACE_PAUSED '3'
#define CLOCK_REMAINING_TIME 'R'
#define CLOCK_ELAPSED_TIME 'E'
#define CLOCK_SEGMENT_REMAINING_TIME 'S'
#define LAPS_REMAINING 'L'

class Race {
  protected:
    char state;
    char previousState;
    bool falseStartEnabled;
    bool falseStartDetected;
    bool startingLights;
    unsigned long penaltyBeginMillis;
    unsigned long penaltyServedMillis;
    unsigned long penaltyTimeMillis;
    void penaltyStart() {
      if (previousState ≡ RACE_INIT) {
        penaltyBeginMillis = millis(); // starting the race
      } else if (previousState ≡ RACE_PAUSED) { // resuming current race
        penaltyBeginMillis = penaltyBeginMillis
                             + (millis() - penaltyBeginMillis)
                             - penaltyServedMillis;
      }
    }
    unsigned long getPenaltyServedMillis() {
      if (falseStartDetected ∧ isStarted()) {
        penaltyServedMillis = millis() - penaltyBeginMillis;
      }
      return penaltyServedMillis;
    }
  public:
    Race() {
      state = RACE_FINISHED;
      previousState = RACE_FINISHED;
      falseStartEnabled = false;
      falseStartDetected = false;
      startingLights = LOW;
      penaltyBeginMillis = 0L;
      penaltyServedMillis = 0L;
      penaltyTimeMillis = 0L;
    }
    void debug() {
      Serial3.print("      Started ? "); Serial3.println(isStarted() ? "yes" : "no");
      Serial3.print("       Paused ? "); Serial3.println(isPaused() ? "yes" : "no");
      Serial3.print("     Finished ? "); Serial3.println(isFinished () ? "yes" : "no");
      Serial3.print("         Init ? "); Serial3.println(isInit() ? "yes" : "no");
      Serial3.print("        state = ");
      switch (state) {
        case RACE_INIT: {
            Serial3.println("Race Init");
            break;
          }
        case RACE_STARTED: {
            Serial3.println("Race Started");
            break;
          }
        case RACE_FINISHED: {
            Serial3.println("Race Finished");
            break;
          }
        case RACE_PAUSED: {
            Serial3.println("Race Paused");
            break;
          }
        default: {
            Serial3.println("unknown");
          }
      }
      Serial3.print("       Served ? "); Serial3.println(isFalseStartPenaltyServed() ? "yes" : "no");
      Serial3.print(" falseStartEnabled = "); Serial3.println(falseStartEnabled ? "yes" : "no");
      Serial3.print(" falseStartDetected = "); Serial3.println(falseStartDetected ? "yes" : "no");
      Serial3.print(" penaltyBeginMillis = "); Serial3.println(penaltyBeginMillis);
      Serial3.print("penaltyServedMillis = "); Serial3.println(getPenaltyServedMillis());
      Serial3.print("  penaltyTimeMillis = "); Serial3.println(penaltyTimeMillis);
      Serial3.print("          now = "); Serial3.println(millis());
    }
    void initFalseStart(byte mode) {
      falseStartEnabled = mode > 7;
      if (falseStartEnabled) { // false start HW enabled
        falseStartDetected = false; // reset false start race "fuse"
        penaltyBeginMillis = 0xFFFFFFFF;
        penaltyServedMillis = 0;
        penaltyTimeMillis = delayMillis[mode - 8];
      }
    }
    void setFalseStartDetected() {
      falseStartDetected = true;
    }
    bool isFalseStartPenaltyServed() {
      return getPenaltyServedMillis() > penaltyTimeMillis;
    }
    bool isFalseStartDetected() {
      return falseStartDetected;
    }
    bool isFalseStartEnabled() {
      return falseStartEnabled;
    }
    bool isStarted() {
      return state ≡ RACE_STARTED;
    }
    bool isPaused() {
      return state ≡ RACE_PAUSED;
    }
    bool isFinished () {
      return state ≡ RACE_FINISHED;
    }
    bool isInit() {
      return state ≡ RACE_INIT;
    }
    bool fromState(char from) {
      return from ≡ previousState;
    }
    void init() {
      previousState = state;
      state = RACE_INIT;
    }
    void start() {
      previousState = state;
      state = RACE_STARTED;
      penaltyStart();
    }
    void pause() {
      previousState = state;
      state = RACE_PAUSED;
    }
    void finish() {
      previousState = state;
      state = RACE_FINISHED;
    }
    void startingLightsOn() {
      startingLights = HIGH;
    }
    void startingLightsOff() {
      startingLights = LOW;
    }
    bool areStartingLightsOff() {
      return startingLights ≡ LOW;
    }
    bool areStartingLightsOn() {
      return startingLights ≡ HIGH;
    }
};

/*****************************************************************************************
   Class Race instantiations
 *****************************************************************************************/
Race race;

/*****************************************************************************************
   Class Lane
 *****************************************************************************************/
class Lane {
  protected:
    volatile unsigned long start;
    volatile unsigned long finish;
    volatile unsigned long now;
    volatile long count;
    volatile bool reported;
    byte lane;
    byte pin;
    byte green;
    byte red;
    bool falseStart;
  public:
    Lane(byte setLane) {
      start = 0L;
      finish = 0L;
      count = -1L;
      lane = setLane - 1;
      pin = laneToRelayMapping[lane];
      green = laneToGreenMapping[lane];
      red = laneToRedMapping[lane];
      reported = true;
      falseStart = false;
    }
    void lapDetected() { // called by ISR, short and sweet
      now = millis();
      if ((now - finish) < laneProtectionTime) {
        return;
      }
      start = finish;
      finish = now;
      count++;
      reported = false;
    }
    void reset() {
      reported = true;
      falseStart = false;
      count = -1L;
    }
    void reportLap() {
      if (¬reported) {
        Serial.print(lapTime[lane]);
        Serial.print(finish - start);
        Serial.println(']');
        reported = true;
      }
      if (race.isFalseStartEnabled()) {
        if (race.isInit() ∧ ¬falseStart ∧ (count ≡ 0)) {
          // false start detected,
          // switching lane off immediately
          powerOff();
          falseStart = true;
          race.setFalseStartDetected(); // burn the race fuse
        }
        // switch power back on after false start penalty served
        if (falseStart ∧ race.isFalseStartPenaltyServed()) {
          falseStart = false; // reset false start lane "fuse"
          powerOn();
        }
      }
    }
    void powerOn() {
      if (¬falseStart) {
        digitalWrite(pin, HIGH);
        digitalWrite(red, LOW);
        digitalWrite(green, HIGH);
      } else {
        digitalWrite(red, HIGH);
        digitalWrite(green, LOW);
      }
    }
    void powerOff() {
      digitalWrite(pin, LOW);
      digitalWrite(red, HIGH);
      digitalWrite(green, LOW);
    }
    bool isFalseStart() {
      return falseStart;
    }
};

/*****************************************************************************************
   Class Lane instantiations
 *****************************************************************************************/
Lane lane1(1);
Lane lane2(2);
Lane lane3(3);
Lane lane4(4);
Lane lane5(5);
Lane lane6(6);

/*****************************************************************************************
   Class Button - external buttons for PC Lap Counter
 *****************************************************************************************/
```

```
class Button {
  protected:
    String button;
    byte pin;
    unsigned int sleep;
    bool reported;
    bool pressed;
    void reportButton() {
      Serial.println(button);
      reported = true;
    }
  public:
    Button(String setButton, byte setPin, unsigned int setSleep) {
      button = setButton;
      pin = setPin;
      sleep = setSleep;
      reported = false;
      pressed = false;
      pinMode(pin, INPUT_PULLUP);
    }
    void isButtonPressed() {
      pressed = ¬digitalRead(pin);
      if (¬reported ∧ pressed) {
        reportButton();
        delay(sleep);
      }
      reported = pressed;
    }
};

/*****************************************************************************************
   Class Button instantiations
 *****************************************************************************************/
Button raceStart(BUTTON_RACE_START,            47, 10); // pin 5 (RJ11 1)
Button raceRestart(BUTTON_RACE_RESTART,        45, 10); // pin 6 (RJ11 2)
Button racePause(BUTTON_RACE_PAUSE,            43, 10); // pin 7 (RJ11 3, RJ11 4 = GND)
Button raceStartPauseRestart(BUTTON_RACE_NEXT, 33, 100); // pin 1 (RJ11 n/c)
//Button powerOff(BUTTON_POWER_OFF, 48);
//Button powerOn(BUTTON_POWER_ON, 49);
//Button endOfRace(BUTTON_END_OF_RACE, 50);
//Button togglePower(BUTTON_TOGGLE_POWER, 51);
//Button toggleYelloFlag(BUTTON_TOGGLE_YELLOW_FLAG, 52);
//Button stopAndGoLane1(BUTTON_STOP_AND_GO_LANE1);
//Button stopAndGoLane2(BUTTON_STOP_AND_GO_LANE2, 23);
//Button stopAndGoLane3(BUTTON_STOP_AND_GO_LANE3", 24);
//Button stopAndGoLane4(BUTTON_STOP_AND_GO_LANE4, 25);
//Button stopAndGoLane5(BUTTON_STOP_AND_GO_LANE5, 26);
//Button stopAndGoLane6(BUTTON_STOP_AND_GO_LANE6, 27);

/*****************************************************************************************
   Class FalseStart - HW solution setup false start enable/disable, detection and penalty
 *****************************************************************************************/
class FalseStart {
  protected:
    void reset() {
      // reset false start flags
      lane1.reset();
      lane2.reset();
      lane3.reset();
      lane4.reset();
      lane5.reset();
      lane6.reset();
    }
  public:
    FalseStart() {
      // empty constructor
    }
    void init() {
      // read pins of 4-bit encoder
      byte mode = ¬digitalRead(FSbit_3) << 3 |
                  ¬digitalRead(FSbit_2) << 2 |
                  ¬digitalRead(FSbit_1) << 1 |
                  ¬digitalRead(FSbit_0);
      race.initFalseStart(mode);
      reset();
    }
};
```

```
/*****************************************************************************
   Class FalseStart instantiations
 *****************************************************************************/
FalseStart falseStart;

/*****************************************************************************
   initializations and configurations of I/O pins
 *****************************************************************************/
void setup() {
  // initialize serial communication
  Serial.begin(serialSpeed);
  while (¬Serial) {
    ; // wait for serial port to connect. Needed for native USB
  }
  Serial3.begin(serial3Speed);
  while (¬Serial3) {
    ; // wait...
  }
  // interrup pins
  pinMode(LANE_1, INPUT_PULLUP);
  pinMode(LANE_2, INPUT_PULLUP);
  pinMode(LANE_3, INPUT_PULLUP);
  pinMode(LANE_4, INPUT_PULLUP);
  pinMode(LANE_5, INPUT_PULLUP);
  pinMode(LANE_6, INPUT_PULLUP);
  // input pins
  pinMode(FSbit_0, INPUT_PULLUP);
  pinMode(FSbit_1, INPUT_PULLUP);
  pinMode(FSbit_2, INPUT_PULLUP);
  pinMode(FSbit_3, INPUT_PULLUP);
  // shake the dust off the relays
  jiggleRelays();
  delay(333);
  // light show
  lightShow(initPattern, sizeof(initPattern));
  delay(333);
  setPowerOn(); // switch all power relays on
}

/*****************************************************************************
   relays initialization - shake the dust off the contacts
 *****************************************************************************/
#define CLICK 20

void jiggleRelays() {
  allRelaysOn();
  delay(CLICK);
  allRelaysOff();
  delay(222);
  allRelaysOn();
  delay(CLICK);
  allRelaysOff();
  delay(111);
  allRelaysOn();
  delay(CLICK);
  allRelaysOff();
  delay(111);
  allRelaysOn();
  delay(CLICK);
  allRelaysOff();
  delay(222);
  allRelaysOn();
  delay(CLICK);
  allRelaysOff();
  delay(444);
  allRelaysOn();
  delay(CLICK);
  allRelaysOff();
  delay(222);
  allRelaysOn();
  delay(CLICK);
  allRelaysOff();
}

/*****************************************************************************
   Class LED
 *****************************************************************************/
class LED {
  protected:
    byte led;
  public:
    LED (byte setLed) {
      led = setLed;
      pinMode(led, OUTPUT);
    }
    void on() {
      digitalWrite(led, true);
    }
    void off() {
      digitalWrite(led, false);
    }
};

LED startFinishLED1(LED_1);
LED startFinishLED2(LED_2);
LED startFinishLED3(LED_3);
LED startFinishLED4(LED_4);
LED startFinishLED5(LED_5);
LED ledGO(LED_GO);
LED ledSTOP(LED_STOP);
LED ledPowerAll(PWR_ALL);

/*****************************************************************************************
   Class RacerStandLED
 *****************************************************************************************/
class RacerStandLED {
  protected:
    byte greenPin;
    byte redPin;
    bool isRed;
    bool isGreen;
    void apply() {
      digitalWrite(greenPin, isGreen);
      digitalWrite(redPin, isRed);
    }
  public:
    RacerStandLED(byte lane) {
      greenPin = laneToGreenMapping[lane - 1];
      redPin = laneToRedMapping[lane - 1];
      pinMode(greenPin, OUTPUT);
      pinMode(redPin, OUTPUT);
    }
    void off() {
      isRed = false;
      isGreen = false;
      apply();
    }
    void red() {
      isRed = true;
      isGreen = false;
      apply();
    }
    void green() {
      isRed = false;
      isGreen = true;
      apply();
    }
    void yellow() {
      isRed = true;
      isGreen = true;
      apply();
    }
};

RacerStandLED racerStandLED1(1);
RacerStandLED racerStandLED2(2);
RacerStandLED racerStandLED3(3);
RacerStandLED racerStandLED4(4);
RacerStandLED racerStandLED5(5);
RacerStandLED racerStandLED6(6);

/*****************************************************************************************
   Class Relay
 *****************************************************************************************/
class Relay {
  protected:
    byte pin;
  public:
    Relay(byte lane) {
      pin = laneToRelayMapping[lane - 1];
      pinMode(pin, OUTPUT);
    }
    void on() {
      digitalWrite(pin, HIGH);
    }
    void off() {
      digitalWrite(pin, LOW);
    }
};

Relay relay1(1);
Relay relay2(2);
Relay relay3(3);
Relay relay4(4);
Relay relay5(5);
Relay relay6(6);

/*****************************************************************************************
   engage/disengage relays
 ****************************************************************************************/
void allRelaysOn() {
  relay1.on();
  relay2.on();
  relay3.on();
  relay4.on();
  relay5.on();
  relay6.on();
}

void allRelaysOff() {
  relay1.off();
  relay2.off();
  relay3.off();
  relay4.off();
  relay5.off();
  relay6.off();
}

void setPowerOn() {
  ledPowerAll.on();
  allRelaysOn();
  setLEDsPowerOn();
}

void setPowerOff() {
  ledPowerAll.off();
  allRelaysOff();
  setLEDsPowerOff();
}

/*****************************************************************************************
   corresponding LEDs pattern for engage/disengage relays
 ****************************************************************************************/
void setLEDsPowerOn() {
  startFinishLED1.off();
  startFinishLED2.off();
  startFinishLED3.off();
  startFinishLED4.off();
  startFinishLED5.off();
  ledGO.on();
  ledSTOP.off();
  setAllRacersGreen();
}

void setLEDsPowerOff() {
  startFinishLED1.on();
  startFinishLED2.on();
  startFinishLED3.on();
  startFinishLED4.on();
  startFinishLED5.on();
  ledGO.off();
  ledSTOP.on();
  setAllRacersRed();
}
```

```
void setAllRacersGreen() {
  racerStandLED1.green();
  racerStandLED2.green();
  racerStandLED3.green();
  racerStandLED4.green();
  racerStandLED5.green();
  racerStandLED6.green();
}

void setAllRacersRed() {
  racerStandLED1.red();
  racerStandLED2.red();
  racerStandLED3.red();
  racerStandLED4.red();
  racerStandLED5.red();
  racerStandLED6.red();
}

void setAllRacersYellow() {
  racerStandLED1.yellow();
  racerStandLED2.yellow();
  racerStandLED3.yellow();
  racerStandLED4.yellow();
  racerStandLED5.yellow();
  racerStandLED6.yellow();
}

void setAllRacersOff() {
  racerStandLED1.off();
  racerStandLED2.off();
  racerStandLED3.off();
  racerStandLED4.off();
  racerStandLED5.off();
  racerStandLED6.off();
}

/******************************************************************************************
   Light Show
 ******************************************************************************************/
void lightShow(const byte pattern[][PATTERN_COLUMNS], int totalSize) {
  // noob note: we're passing a pointer and the size is always sizeof(pattern) = 2!!!
  int numberOfPatterns = totalSize / PATTERN_COLUMNS;
  for (int i = 0; i < numberOfPatterns; i++) {
    pattern[i][1]  ≡ 1 ? ledGO.on() : ledGO.off();
    pattern[i][2]  ≡ 1 ? startFinishLED5.on() : startFinishLED5.off();
    pattern[i][3]  ≡ 1 ? startFinishLED4.on() : startFinishLED4.off();
    pattern[i][4]  ≡ 1 ? startFinishLED3.on() : startFinishLED3.off();
    pattern[i][5]  ≡ 1 ? startFinishLED2.on() : startFinishLED2.off();
    pattern[i][6]  ≡ 1 ? startFinishLED1.on() : startFinishLED1.off();
    pattern[i][7]  ≡ 1 ? racerStandLED6.red() :
    pattern[i][7]  ≡ 2 ? racerStandLED6.green() :
    pattern[i][7]  ≡ 3 ? racerStandLED6.yellow() : racerStandLED6.off();
    pattern[i][8]  ≡ 1 ? racerStandLED5.red() :
    pattern[i][8]  ≡ 2 ? racerStandLED5.green() :
    pattern[i][8]  ≡ 3 ? racerStandLED5.yellow() : racerStandLED5.off();
    pattern[i][9]  ≡ 1 ? racerStandLED4.red() :
    pattern[i][9]  ≡ 2 ? racerStandLED4.green() :
    pattern[i][9]  ≡ 3 ? racerStandLED4.yellow() : racerStandLED4.off();
    pattern[i][10] ≡ 1 ? racerStandLED3.red() :
    pattern[i][10] ≡ 2 ? racerStandLED3.green() :
    pattern[i][10] ≡ 3 ? racerStandLED3.yellow() : racerStandLED3.off();
    pattern[i][11] ≡ 1 ? racerStandLED2.red() :
    pattern[i][11] ≡ 2 ? racerStandLED2.green() :
    pattern[i][11] ≡ 3 ? racerStandLED2.yellow() : racerStandLED2.off();
    pattern[i][12] ≡ 1 ? racerStandLED1.red() :
    pattern[i][12] ≡ 2 ? racerStandLED1.green() :
    pattern[i][12] ≡ 3 ? racerStandLED1.yellow() : racerStandLED1.off();
    delay(pattern[i][0]);
  }
}

/******************************************************************************************
   enable interrupts
 ******************************************************************************************/
void attachAllInterrupts() {
  attachInterrupt(digitalPinToInterrupt(LANE_1), lapDetected1, RISING);
  attachInterrupt(digitalPinToInterrupt(LANE_2), lapDetected2, RISING);
  attachInterrupt(digitalPinToInterrupt(LANE_3), lapDetected3, RISING);
  attachInterrupt(digitalPinToInterrupt(LANE_4), lapDetected4, RISING);
  attachInterrupt(digitalPinToInterrupt(LANE_5), lapDetected5, RISING);
  attachInterrupt(digitalPinToInterrupt(LANE_6), lapDetected6, RISING);
}

/*****************************************************************************************
   disable interrupts
 *****************************************************************************************/
void detachAllInterrupts() {
  detachInterrupt(digitalPinToInterrupt(LANE_1));
  detachInterrupt(digitalPinToInterrupt(LANE_2));
  detachInterrupt(digitalPinToInterrupt(LANE_3));
  detachInterrupt(digitalPinToInterrupt(LANE_4));
  detachInterrupt(digitalPinToInterrupt(LANE_5));
  detachInterrupt(digitalPinToInterrupt(LANE_6));
}

/*****************************************************************************************
   Interrup Service Routines (ISR) definitions
 *****************************************************************************************/
void lapDetected1() {
  lane1.lapDetected();
}
void lapDetected2() {
  lane2.lapDetected();
}
void lapDetected3() {
  lane3.lapDetected();
}
void lapDetected4() {
  lane4.lapDetected();
}
void lapDetected5() {
  lane5.lapDetected();
}
void lapDetected6() {
  lane6.lapDetected();
}

/*****************************************************************************************
   Main loop
 *****************************************************************************************/
void loop() {
  detachAllInterrupts();
  while (Serial3.available()) {
    String command = Serial3.readStringUntil(',');
    if (command ≡ "show") {
      lightShow(showPattern, sizeof(showPattern));
      setLEDsPowerOn();
    }
    if (command ≡ "init") {
      lightShow(initPattern, sizeof(initPattern));
      setLEDsPowerOn();
    }
    if (command ≡ "status") {
      race.debug();
    }
  }
  while (Serial.available()) {
    Serial.readStringUntil('[');
    {
      String output = Serial.readStringUntil(']');
      Serial3.println(output);
      String raceClockState = output.substring(0, 3); // RC#
      // String raceClockTime = output.substring(4, 8); // HH:MM:SS
      if (raceClockState ≡ "RC0") { // Race Clock - Race Setup
        if (race.fromState(RACE_FINISHED)) {
          setPowerOff();
        }
        race.init();
        falseStart.init();
        // } else if (raceClockState == "RC1" && !race.isStarted) { // Race Clock - Race Started
        //   race.start(); // misses the first second
      } else if (raceClockState ≡ "RC2") { // Race Clock - Race Finished
        race.finish();
        startFinishLED1.on();
        startFinishLED2.on();
        startFinishLED3.on();
        startFinishLED4.on();
        startFinishLED5.on();
      } else if (raceClockState ≡ "RC3" ∧ ¬race.isPaused()) { // Race Clock - Race Paused
        race.pause(); // track call immediate, segment end after detection delay
        setAllRacersYellow();
      } else if (output ≡ SL_1_ON) {
        race.startingLightsOn(); // set race starting light state with LED1 only
        startFinishLED1.on();
      } else if (output ≡ SL_1_OFF) {
        race.startingLightsOff(); // set race starting light state with LED1 only
        startFinishLED1.off();
      } else if (output ≡ SL_2_ON) {
        startFinishLED2.on();
      } else if (output ≡ SL_2_OFF) {
        startFinishLED2.off();
      } else if (output ≡ SL_3_ON) {
        startFinishLED3.on();
      } else if (output ≡ SL_3_OFF) {
        startFinishLED3.off();
      } else if (output ≡ SL_4_ON) {
        startFinishLED4.on();
      } else if (output ≡ SL_4_OFF) {
        startFinishLED4.off();
      } else if (output ≡ SL_5_ON) {
        startFinishLED5.on();
      } else if (output ≡ SL_5_OFF) {
        startFinishLED5.off();
      } else if (output ≡ GO_ON) { // race start
        race.start();
        ledGO.on();
        setAllRacersGreen();
      } else if (output ≡ GO_OFF) { // track call, segment or heat end
        race.pause();
        ledGO.off();
      } else if (output ≡ STOP_ON) {
        ledSTOP.on();
        if (race.isPaused() ∧ race.fromState(RACE_STARTED)) { // blink
          startFinishLED1.off();
          startFinishLED2.on();
          startFinishLED3.off();
          startFinishLED4.on();
          startFinishLED5.off();
          setAllRacersYellow();
        }
      } else if (output ≡ STOP_OFF) {
        ledSTOP.off();
        // flickers when race is continued (track or segment)
        if (race.isPaused() ∧
            race.fromState(RACE_STARTED) ∧
            race.areStartingLightsOff()) { // blink
          startFinishLED1.on();
          startFinishLED2.off();
          startFinishLED3.on();
          startFinishLED4.off();
          startFinishLED5.on();
          setAllRacersOff();
        }
      } else if (output ≡ PWR_ON) {
        ledPowerAll.on();
        setAllRacersYellow();
        if (race.isFinished()) {
          setPowerOn();
        }
      } else if (output ≡ PWR_OFF) {
        ledPowerAll.off();
        if (race.isFinished()) {
          setPowerOff();
        }
      } else if (output ≡ PWR_1_ON) {
        lane1.powerOn();
      } else if (output ≡ PWR_1_OFF) {
        lane1.powerOff();
      } else if (output ≡ PWR_2_ON) {
        lane2.powerOn();
      } else if (output ≡ PWR_2_OFF) {
        lane2.powerOff();
      } else if (output ≡ PWR_3_ON) {
        lane3.powerOn();
      } else if (output ≡ PWR_3_OFF) {
        lane3.powerOff();
      } else if (output ≡ PWR_4_ON) {
        lane4.powerOn();
      } else if (output ≡ PWR_4_OFF) {
        lane4.powerOff();
      } else if (output ≡ PWR_5_ON) {
        lane5.powerOn();
      } else if (output ≡ PWR_5_OFF) {
        lane5.powerOff();
      } else if (output ≡ PWR_6_ON) {
        lane6.powerOn();
      } else if (output ≡ PWR_6_OFF) {
        lane6.powerOff();
      } else if (raceClockState ≡ "DEB") {
        race.debug();
      }
    }
  }
  /** report lap if necessary */
  lane1.reportLap();
  lane2.reportLap();
  lane3.reportLap();
  lane4.reportLap();
  lane5.reportLap();
  lane6.reportLap();
  /** any buttons pressed */
  raceStart.isButtonPressed();
  raceRestart.isButtonPressed();
  racePause.isButtonPressed();
  delay(3);
  attachAllInterrupts();
}
```