```
/*******************************************************************************
    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:
    - 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 laneDetectionBlackoutPeriod = 500L;
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
};

/*******************************************************************************
    Symbol Definitions
 *******************************************************************************/
```

```c
     #define ON HIGH
80   #define OFF LOW

     /*************************************************************************************
        Arduono Button Press Messages
        ***********************************************************************************/
85   #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]"
90   #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]"
95   #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]"
100
     /*************************************************************************************
        Pin Naming
        ***********************************************************************************/
     // lane to interrup pin mapping
105  #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]
110  #define LANE_6 laneToInterrupMapping[5]

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

     #define LED_DSR1 41
     #define LED_DSG1 44
120  #define LED_DSR2 42
     #define LED_DSG2 46
     #define LED_DSR3 40
     #define LED_DSG3 38
     #define LED_DSR4 36
125  #define LED_DSG4 34
     #define LED_DSR5 32
     #define LED_DSG5 39
     #define LED_DSR6 37
     #define LED_DSG6 35
130
     #define LED_STOP 22
     #define LED_CAUTION 24
     #define LED_GO 26

135  // 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
140  #define PWR_4   laneToRelayMapping[3] //  9
     #define PWR_5   laneToRelayMapping[4] //  7
     #define PWR_6   laneToRelayMapping[5] //  5

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

     /*************************************************************************************
150     PC Lap Counter Messages
        ***********************************************************************************/
     #define SL_1_ON   "SL011"
     #define SL_1_OFF "SL010"
     #define SL_2_ON   "SL021"
155  #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"
160     #define SL_5_ON  "SL051"
        #define SL_5_OFF "SL050"


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


170     #define PWR_ON     "PW001"
        #define PWR_OFF    "PW000"
        #define PWR_1_ON   "PW011"
        #define PWR_1_OFF "PW010"
        #define PWR_2_ON   "PW021"
175     #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"
180     #define PWR_5_ON   "PW051"
        #define PWR_5_OFF "PW050"
        #define PWR_6_ON   "PW061"
        #define PWR_6_OFF "PW060"


185     /*************************************************************************************
            Class Race
            *************************************************************************************/
        #define RACE_INIT '0'
        #define RACE_STARTED '1'
190     #define RACE_FINISHED '2'
        #define RACE_PAUSED '3'
        #define CLOCK_REMAINING_TIME 'R'
        #define CLOCK_ELAPSED_TIME 'E'
        #define CLOCK_SEGMENT_REMAINING_TIME 'S'
195     #define LAPS_REMAINING 'L'


        class Race {
          protected:
            char state;
200         char previousState;
            bool falseStartEnabled;
            bool falseStartDetected;
            bool startingLights;
            unsigned long penaltyBeginMillis;
205         unsigned long penaltyServedMillis;
            unsigned long penaltyTimeMillis;
            void penaltyStart() {
              if (previousState ≡ RACE_INIT) {
                penaltyBeginMillis = millis(); // starting the race
210           } else if (previousState ≡ RACE_PAUSED) { // resuming current race
                penaltyBeginMillis = penaltyBeginMillis
                                     + (millis() − penaltyBeginMillis)
                                     − penaltyServedMillis;
              }
215         }
            unsigned long getPenaltyServedMillis() {
              if (falseStartDetected ∧ isStarted()) {
                penaltyServedMillis = millis() − penaltyBeginMillis;
              }
220           return penaltyServedMillis;
            }
          public:
            Race() {
              state = RACE_FINISHED;
225           previousState = RACE_FINISHED;
              falseStartEnabled = false;
              falseStartDetected = false;
              startingLights = OFF;
              penaltyBeginMillis = 0L;
230           penaltyServedMillis = 0L;
              penaltyTimeMillis = 0L;
            }
            void debug() {
              Serial3.print("     Started ? "); Serial3.println(isStarted() ? "yes" : "no");
```

```
235          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) {
240            case RACE_INIT: {
                   Serial3.println("Race Init");
                   break;
               }
               case RACE_STARTED: {
245                Serial3.println("Race Started");
                   break;
               }
               case RACE_FINISHED: {
                   Serial3.println("Race Finished");
250                break;
               }
               case RACE_PAUSED: {
                   Serial3.println("Race Paused");
                   break;
255            }
               default: {
                   Serial3.println("unknown");
               }
           }
260        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());
265        Serial3.print(" penaltyTimeMillis = "); Serial3.println(penaltyTimeMillis);
           Serial3.print("          now = "); Serial3.println(millis());
       }
       void initFalseStart(byte mode) {
           falseStartEnabled = mode > 7;
270        if (falseStartEnabled) { // false start HW enabled
             falseStartDetected = false; // reset false start race "fuse"
             penaltyBeginMillis = 0xFFFFFFFF;
             penaltyServedMillis = 0;
             penaltyTimeMillis = delayMillis[mode - 8];
275        }
       }
       void setFalseStartDetected() {
         falseStartDetected = true;
       }
280    bool isFalseStartPenaltyServed() {
         return getPenaltyServedMillis() > penaltyTimeMillis;
       }
       bool isFalseStartDetected() {
         return falseStartDetected;
285    }
       bool isFalseStartEnabled() {
         return falseStartEnabled;
       }
       bool isStarted() {
290      return state == RACE_STARTED;
       }
       bool isPaused() {
         return state == RACE_PAUSED;
       }
295    bool isFinished () {
         return state == RACE_FINISHED;
       }
       bool isInit() {
         return state == RACE_INIT;
300    }
       bool fromState(char from) {
         return from == previousState;
       }
       void init() {
305      previousState = state;
         state = RACE_INIT;
       }
       void start() {
         previousState = state;
310      state = RACE_STARTED;
         penaltyStart();
       }
```

```
            void pause() {
              previousState = state;
315           state = RACE_PAUSED;
            }
            void finish() {
              previousState = state;
              state = RACE_FINISHED;
320         }
            void setStartingLights(bool setOn) {
              startingLights = setOn;
            }
            bool areStartingLights(bool setOn) {
325           return startingLights ≡ setOn;
            }
      };

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

      /*******************************************************************************
335       Class Lane
      *******************************************************************************/
      class Lane {
        protected:
          volatile unsigned long start;
340       volatile unsigned long finish;
          volatile unsigned long now;
          volatile long count;
          volatile bool reported;
          byte lane;
345       byte pin;
          byte green;
          byte red;
          bool falseStart;
        public:
350       Lane(byte setLane) {
            start = 0L;
            finish = 0L;
            count = −1L;
            lane = setLane − 1;
355         pin = laneToRelayMapping[lane];
            green = laneToGreenMapping[lane];
            red = laneToRedMapping[lane];
            reported = true;
            falseStart = false;
360       }
          void lapDetected() { // called by ISR, short and sweet
            now = millis();
            if ((now − finish) < laneDetectionBlackoutPeriod) {
              return;
365         }
            start = finish;
            finish = now;
            count++;
            reported = false;
370       }
          void reset() {
            reported = true;
            falseStart = false;
            count = −1L;
375       }
          void reportLap() {
            if (¬reported) {
              Serial.print(lapTime[lane]);
              Serial.print(finish − start);
380           Serial.println(']');
              reported = true;
            }
            if (race.isFalseStartEnabled()) {
              if (race.isInit() ∧ ¬falseStart ∧ (count ≡ 0)) {
385             // false start detected,
                // switching lane off immediately
                powerOff();
                falseStart = true;
                race.setFalseStartDetected(); // burn the race fuse
390           }
```

```
                // switch power back on after false start penalty served
                if (falseStart ∧ race.isFalseStartPenaltyServed()) {
                  falseStart = false; // reset false start lane "fuse"
                  powerOn();
395             }
            }
        }
        void powerOn() {
            if (¬falseStart) {
400             digitalWrite(pin, HIGH);
                digitalWrite(red, LOW);
                digitalWrite(green, HIGH);
            } else {
                digitalWrite(red, HIGH);
405             digitalWrite(green, HIGH);
            }
        }
        void powerOff() {
            digitalWrite(pin, LOW);
410         digitalWrite(red, HIGH);
            digitalWrite(green, LOW);
        }
        bool isFalseStart() {
            return falseStart;
415     }
    };

    /*******************************************************************************
       Class Lane instantiations
420  *******************************************************************************/
    Lane lane1(1);
    Lane lane2(2);
    Lane lane3(3);
    Lane lane4(4);
425 Lane lane5(5);
    Lane lane6(6);

    /*******************************************************************************
       Class Button - external buttons for PC Lap Counter
430  *******************************************************************************/
    class Button {
      protected:
        String button;
        byte pin;
435     unsigned int sleep;
        bool reported;
        bool pressed;
        void reportButton() {
            Serial.println(button);
440         reported = true;
        }
      public:
        Button(String setButton, byte setPin, unsigned int setSleep) {
            button = setButton;
445         pin = setPin;
            sleep = setSleep;
            reported = false;
            pressed = false;
            pinMode(pin, INPUT_PULLUP);
450     }
        void isButtonPressed() {
            pressed = ¬digitalRead(pin);
            if (¬reported ∧ pressed) {
                reportButton();
455             delay(sleep);
            }
            reported = pressed;
        }
    };
460
    /*******************************************************************************
       Class Button instantiations
     *******************************************************************************/
    Button raceStart(BUTTON_RACE_START,            47, 10); // pin 5 (RJ11 1)
465 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);
470   //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);
475   //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);


480   /*********************************************************************************
         Class FalseStart - HW solution setup false start enable/disable, detection and penalty
      *********************************************************************************/
      class FalseStart {
        protected:
485       void reset() {
            // reset false start flags
            lane1.reset();
            lane2.reset();
            lane3.reset();
490         lane4.reset();
            lane5.reset();
            lane6.reset();
          }
        public:
495       FalseStart() {
            // empty constructor
          }
          void init() {
            // read pins of 4-bit encoder
500         byte mode = ¬digitalRead(FSbit_3) << 3 |
                         ¬digitalRead(FSbit_2) << 2 |
                         ¬digitalRead(FSbit_1) << 1 |
                         ¬digitalRead(FSbit_0);
            race.initFalseStart(mode);
505         reset();
          }
      };

      /*********************************************************************************
510      Class FalseStart instantiations
      *********************************************************************************/
      FalseStart falseStart;

      /*********************************************************************************
515      initializations and configurations of I/O pins
      *********************************************************************************/
      void setup() {
        // interrup pins
        pinMode(LANE_1, INPUT_PULLUP);
520     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);
525     // input pins
        pinMode(FSbit_0, INPUT_PULLUP);
        pinMode(FSbit_1, INPUT_PULLUP);
        pinMode(FSbit_2, INPUT_PULLUP);
        pinMode(FSbit_3, INPUT_PULLUP);
530     // shake the dust off the relays
        jiggleRelays();
        delay(1000);
        // light show
        lightShow(3);
535     delay(1000);
        // initialize globals
        setPowerOn(); // switch all power relays on
        // all defined, ready to read/write from/to serial port
        Serial.begin(serialSpeed);
540     while (¬Serial) {
          ; // wait for serial port to connect. Needed for native USB
        }
        Serial3.begin(serial3Speed);
        while (¬Serial3) {
545       ; // wait...
        }
```

```
    }

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

    void jiggleRelays() {
555   allRelaysOn();
      delay(CLICK);
      allRelaysOff();
      delay(222);
      allRelaysOn();
560   delay(CLICK);
      allRelaysOff();
      delay(111);
      allRelaysOn();
      delay(CLICK);
565   allRelaysOff();
      delay(111);
      allRelaysOn();
      delay(CLICK);
      allRelaysOff();
570   delay(222);
      allRelaysOn();
      delay(CLICK);
      allRelaysOff();
      delay(444);
575   allRelaysOn();
      delay(CLICK);
      allRelaysOff();
      delay(222);
      allRelaysOn();
580   delay(CLICK);
      allRelaysOff();
    }


    /********************************************************************************
585     Class LED
    ********************************************************************************/
    class LED {
      protected:
        byte led;
590   public:
        LED (byte setLed) {
          led = setLed;
          pinMode(led, OUTPUT);
        }
595     void on() {
          digitalWrite(led, true);
        }
        void off() {
          digitalWrite(led, false);
600     }
    };

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

    /********************************************************************************
      Class RacerStandLED
    ********************************************************************************/
615 class RacerStandLED {
      protected:
        byte greenPin;
        byte redPin;
        bool isRed;
620     bool isGreen;
        void apply() {
          digitalWrite(greenPin, isGreen);
          digitalWrite(redPin, isRed);
        }
```

```
625      public:
           RacerStandLED(byte lane) {
             greenPin = laneToGreenMapping[lane - 1];
             redPin = laneToRedMapping[lane - 1];
             pinMode(greenPin, OUTPUT);
630          pinMode(redPin, OUTPUT);
           }
           void off() {
             isRed = false;
             isGreen = false;
635          apply();
           }
           void red() {
             isRed = true;
             isGreen = false;
640          apply();
           }
           void green() {
             isRed = false;
             isGreen = true;
645          apply();
           }
           void yellow() {
             isRed = true;
             isGreen = true;
650          apply();
           }
       };

     RacerStandLED racerStandLED1(1);
655  RacerStandLED racerStandLED2(2);
     RacerStandLED racerStandLED3(3);
     RacerStandLED racerStandLED4(4);
     RacerStandLED racerStandLED5(5);
     RacerStandLED racerStandLED6(6);
660
     /******************************************************************************
         Class Relay
      ******************************************************************************/
     class Relay {
665    protected:
         byte pin;
       public:
         Relay(byte lane) {
           pin = laneToRelayMapping[lane - 1];
670        pinMode(pin, OUTPUT);
         }
         void on() {
           digitalWrite(pin, HIGH);
         }
675      void off() {
           digitalWrite(pin, LOW);
         }
     };

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

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

     void allRelaysOff() {
700    relay1.off();
       relay2.off();
       relay3.off();
```

```
        relay4.off();
        relay5.off();
705     relay6.off();
    }

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

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

720  /*******************************************************************************
        corresponding LEDs pattern for engage/disengage relays
        *******************************************************************************/
    void setLEDsPowerOn() {
        startFinishLED1.off();
725     startFinishLED2.off();
        startFinishLED3.off();
        startFinishLED4.off();
        startFinishLED5.off();
        ledGO.on();
730     ledSTOP.off();
        setAllRacersGreen();
    }

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

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

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

    void setAllRacersYellow() {
        racerStandLED1.yellow();
765     racerStandLED2.yellow();
        racerStandLED3.yellow();
        racerStandLED4.yellow();
        racerStandLED5.yellow();
        racerStandLED6.yellow();
770  }

    void setAllRacersOff() {
        racerStandLED1.off();
        racerStandLED2.off();
775     racerStandLED3.off();
        racerStandLED4.off();
        racerStandLED5.off();
        racerStandLED6.off();
    }
780
```

```
     /****************************************************************************************
       Light Show

       patter positions:
785    Duration [ms], Go, SF5, SF4, SF3, SF2, SF1, R6, R5, R4, R3, R2, R1 (STOP & CAUTION n/c)
       0 = off
       1 = on (red)
       2 = green (only for R1..R6)
       3 = yellow (only for R1..R6)
790    ****************************************************************************************/
     const byte pattern[][13] = {
       {150, 1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 3},
       {150, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
       {250, 1, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2},
795    {100, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
       {100, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
       {100, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0},
       {100, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0},
       {100, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0},
800    {100, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0},
       {100, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0},
       {100, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0},
       {100, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0},
       {100, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0},
805    {100, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0},
       {100, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},

       {100, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
       {100, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1},
810    {100, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1},
       {100, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1},
       {100, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1},
       {100, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1},
       {100, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1},
815    {100, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1},
       {100, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},
       {100, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1},
       {255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

820    { 50, 1, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2},
       { 50, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
       { 50, 1, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2},
       { 50, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
       { 50, 1, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2},
825    { 50, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
       { 50, 1, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2},
       { 50, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
       { 50, 1, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2},
       { 50, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
830    { 50, 1, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2},
       { 50, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
       { 50, 1, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2},
       {255, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
     };

835
     #define NUMBER_OF_PATTERNS sizeof(pattern)/sizeof(pattern[0])

     void lightShow(int repetitions) {
       for (int i = 0; i < repetitions; i++) {
840      for (int j = 0; j < NUMBER_OF_PATTERNS; j++) {
           pattern[j][1]  ≡ 1 ? ledGO.on() : ledGO.off();
           pattern[j][2]  ≡ 1 ? startFinishLED5.on() : startFinishLED5.off();
           pattern[j][3]  ≡ 1 ? startFinishLED4.on() : startFinishLED4.off();
           pattern[j][4]  ≡ 1 ? startFinishLED3.on() : startFinishLED3.off();
845        pattern[j][5]  ≡ 1 ? startFinishLED2.on() : startFinishLED2.off();
           pattern[j][6]  ≡ 1 ? startFinishLED1.on() : startFinishLED1.off();
           pattern[j][7]  ≡ 1 ? racerStandLED6.red() :
           pattern[j][7]  ≡ 2 ? racerStandLED6.green() :
           pattern[j][7]  ≡ 3 ? racerStandLED6.yellow() : racerStandLED6.off();
850        pattern[j][8]  ≡ 1 ? racerStandLED5.red() :
           pattern[j][8]  ≡ 2 ? racerStandLED5.green() :
           pattern[j][8]  ≡ 3 ? racerStandLED5.yellow() : racerStandLED5.off();
           pattern[j][9]  ≡ 1 ? racerStandLED4.red() :
           pattern[j][9]  ≡ 2 ? racerStandLED4.green() :
855        pattern[j][9]  ≡ 3 ? racerStandLED4.yellow() : racerStandLED4.off();
           pattern[j][10] ≡ 1 ? racerStandLED3.red() :
           pattern[j][10] ≡ 2 ? racerStandLED3.green() :
           pattern[j][10] ≡ 3 ? racerStandLED3.yellow() : racerStandLED3.off();
```

```
               pattern[j][11] ≡ 1 ? racerStandLED2.red() :
860            pattern[j][11] ≡ 2 ? racerStandLED2.green() :
               pattern[j][11] ≡ 3 ? racerStandLED2.yellow() : racerStandLED2.off();
               pattern[j][12] ≡ 1 ? racerStandLED1.red() :
               pattern[j][12] ≡ 2 ? racerStandLED1.green() :
               pattern[j][12] ≡ 3 ? racerStandLED1.yellow() : racerStandLED1.off();
865            delay(pattern[j][0]);
           }
         }
     }


870 /******************************************************************************
       enable interrupts
     ******************************************************************************/
     void attachAllInterrupts() {
       attachInterrupt(digitalPinToInterrupt(LANE_1), lapDetected1, RISING);
875    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);
880 }

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

    /******************************************************************************
895    Interrup Service Routines (ISR) definitions
     ******************************************************************************/
     void lapDetected1() {
       lane1.lapDetected();
     }
900 void lapDetected2() {
       lane2.lapDetected();
     }
     void lapDetected3() {
       lane3.lapDetected();
905 }
     void lapDetected4() {
       lane4.lapDetected();
     }
     void lapDetected5() {
910    lane5.lapDetected();
     }
     void lapDetected6() {
       lane6.lapDetected();
     }
915
    /******************************************************************************
       Main loop
     ******************************************************************************/
     void loop() {
920    detachAllInterrupts();
       while (Serial.available()) {
         Serial.readStringUntil('[');
         {
           String output = Serial.readStringUntil(']');
925        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)) {
930            setPowerOff();
             }
             race.init();
             falseStart.init();
             // } else if (raceClockState == "RC1" && !race.isStarted) { // Race Clock – Race Started
935          //   race.start(); // misses the first second
           } else if (raceClockState ≡ "RC2") { // Race Clock – Race Finished
```

```
              race.finish();
              startFinishLED1.on();
              startFinishLED2.on();
940           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
945           setAllRacersYellow();
          } else if (output ≡ SL_1_ON) {
              race.setStartingLights(ON); // set race starting light state with LED1 only
              startFinishLED1.on();
          } else if (output ≡ SL_1_OFF) {
950           race.setStartingLights(OFF); // set race starting light state with LED1 only
              startFinishLED1.off();
          } else if (output ≡ SL_2_ON) {
              startFinishLED2.on();
          } else if (output ≡ SL_2_OFF) {
955           startFinishLED2.off();
          } else if (output ≡ SL_3_ON) {
              startFinishLED3.on();
          } else if (output ≡ SL_3_OFF) {
              startFinishLED3.off();
960       } else if (output ≡ SL_4_ON) {
              startFinishLED4.on();
          } else if (output ≡ SL_4_OFF) {
              startFinishLED4.off();
          } else if (output ≡ SL_5_ON) {
965           startFinishLED5.on();
          } else if (output ≡ SL_5_OFF) {
              startFinishLED5.off();
          } else if (output ≡ GO_ON) { // race start
              race.start();
970           ledGO.on();
              setAllRacersGreen();
          } else if (output ≡ GO_OFF) { // track call, segment or heat end
              race.pause();
              ledGO.off();
975       } else if (output ≡ STOP_ON) {
              ledSTOP.on();
              if (race.isPaused() ∧ race.fromState(RACE_STARTED)) { // blink
                startFinishLED1.off();
                startFinishLED2.on();
980             startFinishLED3.off();
                startFinishLED4.on();
                startFinishLED5.off();
                setAllRacersYellow();
              }
985       } else if (output ≡ STOP_OFF) {
              ledSTOP.off();
              // flickers when race is continued (track or segment)
              if (race.isPaused() ∧
                  race.fromState(RACE_STARTED) ∧
990               race.areStartingLights(OFF)) { // blink
                startFinishLED1.on();
                startFinishLED2.off();
                startFinishLED3.on();
                startFinishLED4.off();
995             startFinishLED5.on();
                setAllRacersOff();
              }
          } else if (output ≡ PWR_ON) {
              ledPowerAll.on();
1000          setAllRacersYellow();
              if (race.isFinished()) {
                setPowerOn();
              }
          } else if (output ≡ PWR_OFF) {
1005          ledPowerAll.off();
              if (race.isFinished()) {
                setPowerOff();
              }
          } else if (output ≡ PWR_1_ON) {
1010          lane1.powerOn();
          } else if (output ≡ PWR_1_OFF) {
              lane1.powerOff();
          } else if (output ≡ PWR_2_ON) {
              lane2.powerOn();
```

```
1015            } else if (output ≡ PWR_2_OFF) {
                   lane2.powerOff();
               } else if (output ≡ PWR_3_ON) {
                   lane3.powerOn();
               } else if (output ≡ PWR_3_OFF) {
1020               lane3.powerOff();
               } else if (output ≡ PWR_4_ON) {
                   lane4.powerOn();
               } else if (output ≡ PWR_4_OFF) {
                   lane4.powerOff();
1025           } else if (output ≡ PWR_5_ON) {
                   lane5.powerOn();
               } else if (output ≡ PWR_5_OFF) {
                   lane5.powerOff();
               } else if (output ≡ PWR_6_ON) {
1030               lane6.powerOn();
               } else if (output ≡ PWR_6_OFF) {
                   lane6.powerOff();
               } else if (raceClockState ≡ "DEB") {
                   race.debug();
1035           }
           }
       }
       /** report lap if necessary */
       lane1.reportLap();
1040   lane2.reportLap();
       lane3.reportLap();
       lane4.reportLap();
       lane5.reportLap();
       lane6.reportLap();
1045   /** any buttons pressed */
       raceStart.isButtonPressed();
       raceRestart.isButtonPressed();
       racePause.isButtonPressed();
       delay(3);
1050   attachAllInterrupts();
   }
```