```
/*****************************************************************************************
   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
   - void startLights(byte pattern): get them patterns figured out

   Revision History
   __________ ____________________ _______________________________________________________
   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;
const long serial3Speed = 19200;
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
 *****************************************************************************************/
#define ON HIGH
#define OFF LOW
```

```
/*************************************************************************************
   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 = OFF;
      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 setStartingLights(bool setOn) {
      startingLights = setOn;
    }
    bool areStartingLights(bool setOn) {
      return startingLights ≡ setOn;
    }
};

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

/*****************************************************************************************
```

```
    Class Lane
  ****************************************************************************************/
class Lane {
  protected:
    volatile unsigned long start;
    volatile unsigned long finish;
    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
      start = finish;
      finish = millis();
      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, HIGH);
      }
    }
    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("[BT01]",   47, 10); // pin 5 (RJ11 1)
Button raceRestart("[BT02]", 45, 10); // pin 6 (RJ11 2)
Button racePause("[BT03]",   43, 10); // pin 7 (RJ11 3, RJ11 4 = GND)
//Button raceStartPauseRestart("[BT04]", 43, 100);
//Button powerOff("[BT05]", 48);
//Button powerOn("[BT06]", 49);
//Button endOfRace("[BT07]", 50);
//Button togglePower("[BT08]", 51);
//Button toggleYelloFlag("[BT09]", 52);
//Button stopAndGoLane1("[SG01]", 22);
//Button stopAndGoLane2("[SG02]", 23);
//Button stopAndGoLane3("[SG03]", 24);
//Button stopAndGoLane4("[SG04]", 25);
//Button stopAndGoLane5("[SG05]", 26);
//Button stopAndGoLane6("[SG06]", 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() {
  // 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);
  // output pins
  pinMode(LED_1, OUTPUT);
  pinMode(LED_2, OUTPUT);
  pinMode(LED_3, OUTPUT);
  pinMode(LED_4, OUTPUT);
  pinMode(LED_5, OUTPUT);
  pinMode(LED_GO, OUTPUT);
  pinMode(LED_STOP, OUTPUT);
  //  pinMode(LED_CAUTION, OUTPUT);
  pinMode(PWR_ALL, OUTPUT);
  pinMode(PWR_1, OUTPUT);
  pinMode(PWR_2, OUTPUT);
  pinMode(PWR_3, OUTPUT);
  pinMode(PWR_4, OUTPUT);
  pinMode(PWR_5, OUTPUT);
  pinMode(PWR_6, OUTPUT);
  // plugin box
  pinMode(LED_DSR1, OUTPUT);
  pinMode(LED_DSR2, OUTPUT);
  pinMode(LED_DSR3, OUTPUT);
  pinMode(LED_DSR4, OUTPUT);
  pinMode(LED_DSR5, OUTPUT);
  pinMode(LED_DSR6, OUTPUT);
  pinMode(LED_DSG1, OUTPUT);
  pinMode(LED_DSG2, OUTPUT);
  pinMode(LED_DSG3, OUTPUT);
  pinMode(LED_DSG4, OUTPUT);
  pinMode(LED_DSG5, OUTPUT);
  pinMode(LED_DSG6, OUTPUT);
  // turn all LEDs off
  digitalWrite(LED_1, LOW);
  digitalWrite(LED_2, LOW);
  digitalWrite(LED_3, LOW);
  digitalWrite(LED_4, LOW);
  digitalWrite(LED_5, LOW);
  digitalWrite(LED_GO, LOW);
  digitalWrite(LED_STOP, LOW);
  //  digitalWrite(LED_CAUTION, LOW);
  digitalWrite(LED_DSR1, LOW);
  digitalWrite(LED_DSR2, LOW);
  digitalWrite(LED_DSR3, LOW);
  digitalWrite(LED_DSR4, LOW);
  digitalWrite(LED_DSR5, LOW);
  digitalWrite(LED_DSR6, LOW);
  digitalWrite(LED_DSG1, LOW);
  digitalWrite(LED_DSG2, LOW);
  digitalWrite(LED_DSG3, LOW);
  digitalWrite(LED_DSG4, LOW);
  digitalWrite(LED_DSG5, LOW);
  digitalWrite(LED_DSG6, LOW);
  digitalWrite(PWR_ALL, LOW);
  digitalWrite(PWR_1, HIGH);
  digitalWrite(PWR_2, HIGH);
  digitalWrite(PWR_3, HIGH);
  digitalWrite(PWR_4, HIGH);
  digitalWrite(PWR_5, HIGH);
  digitalWrite(PWR_6, HIGH);
  // shake the dust off the relays
  jiggleRelays();
  delay(1000);
  // initialize globals
  setPower(ON); // switch all power relays on
  // all defined, ready to read/write from/to serial port
  Serial.begin(serialSpeed);
  while (¬Serial) {
    ; // wait for serial port to connect. Needed for native USB
  }
  Serial3.begin(serial3Speed);
  while (¬Serial3) {
    ; // wait..
  }
}

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

void jiggleRelays() {
  setPower(ON);
  delay(CLICK);
  setPower(OFF);
  delay(222);
  setPower(ON);
  delay(CLICK);
  setPower(OFF);
  delay(111);
  setPower(ON);
  delay(CLICK);
  setPower(OFF);
  delay(111);
  setPower(ON);
  delay(CLICK);
  setPower(OFF);
  delay(222);
  setPower(ON);
  delay(CLICK);
  setPower(OFF);
  delay(444);
  setPower(ON);
  delay(CLICK);
  setPower(OFF);
  delay(222);
  setPower(ON);
  delay(CLICK);
  setPower(OFF);
}

/*************************************************************************************
   engage/disengage relays
 *************************************************************************************/
void setPower(bool setOn) {
  digitalWrite(PWR_ALL, setOn);
  digitalWrite(PWR_1, setOn);
  digitalWrite(PWR_2, setOn);
  digitalWrite(PWR_3, setOn);
  digitalWrite(PWR_4, setOn);
  digitalWrite(PWR_5, setOn);
  digitalWrite(PWR_6, setOn);
  relayLEDsOn(setOn);
}
```

```
/*****************************************************************************
   corresponding LEDs pattern for engage/disengage relays
 *****************************************************************************/
void relayLEDsOn(bool setOn) {
  digitalWrite(LED_1, ¬setOn);
  digitalWrite(LED_2, ¬setOn);
  digitalWrite(LED_3, ¬setOn);
  digitalWrite(LED_4, ¬setOn);
  digitalWrite(LED_5, ¬setOn);
  digitalWrite(LED_GO, setOn);
  digitalWrite(LED_STOP, ¬setOn);
  relayLEDsGreen(setOn);
  relayLEDsRed(¬setOn);
}

void relayLEDsGreen(bool setOn) {
  digitalWrite(LED_DSG1, setOn);
  digitalWrite(LED_DSG2, setOn);
  digitalWrite(LED_DSG3, setOn);
  digitalWrite(LED_DSG4, setOn);
  digitalWrite(LED_DSG5, setOn);
  digitalWrite(LED_DSG6, setOn);
}

void relayLEDsRed(bool setOn) {
  digitalWrite(LED_DSR1, setOn);
  digitalWrite(LED_DSR2, setOn);
  digitalWrite(LED_DSR3, setOn);
  digitalWrite(LED_DSR4, setOn);
  digitalWrite(LED_DSR5, setOn);
  digitalWrite(LED_DSR6, setOn);
}

/*****************************************************************************
   yellow (red & gree) on/off
 *****************************************************************************/
void yellowLEDs(bool setOn) {
  relayLEDsGreen(setOn);
  relayLEDsRed(setOn);
}

/*****************************************************************************
   Start/Finish, Go and Stop LEDs
 *****************************************************************************/
void setLED1(bool setOn) {
  digitalWrite(LED_1, setOn);
}

void setLED2(bool setOn) {
  digitalWrite(LED_2, setOn);
}

void setLED3(bool setOn) {
  digitalWrite(LED_3, setOn);
}

void setLED4(bool setOn) {
  digitalWrite(LED_4, setOn);
}

void setLED5(bool setOn) {
  digitalWrite(LED_5, setOn);
}

void setGO(bool setOn) {
  digitalWrite(LED_GO, setOn);
}

void setSTOP(bool setOn) {
  digitalWrite(LED_STOP, setOn);
}

void setALL(bool setOn) {
  digitalWrite(PWR_ALL, setOn);
}

/*****************************************************************************
   start light pattern switcher
```

```
#define OOOOI  1
#define OOOIO  2
#define OOIOO  4
#define OIOOO  8
#define IOOOO 16
void startLights(byte pattern) {
  digitalWrite(LED_1, pattern & OOOOI);
  digitalWrite(LED_2, pattern & OOOIO);
  digitalWrite(LED_3, pattern & OOIOO);
  digitalWrite(LED_4, pattern & OIOOO);
  digitalWrite(LED_5, pattern & IOOOO);
}
 ***************************************************************************************/

/***************************************************************************************
   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 (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)) {
          setPower(OFF);
        }
        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();
          setLED1(ON);
          setLED2(ON);
          setLED3(ON);
          setLED4(ON);
          setLED5(ON);
        } else if (raceClockState ≡ "RC3" ∧ ¬race.isPaused()) { // Race Clock - Race Paused
          race.pause(); // track call immediate, segment end after detection delay
          yellowLEDs(ON);
        } else if (output ≡ SL_1_ON) {
          race.setStartingLights(ON); // set race starting light state with LED1 only
          setLED1(ON);
        } else if (output ≡ SL_1_OFF) {
          race.setStartingLights(OFF); // set race starting light state with LED1 only
          setLED1(OFF);
        } else if (output ≡ SL_2_ON) {
          setLED2(ON);
        } else if (output ≡ SL_2_OFF) {
          setLED2(OFF);
        } else if (output ≡ SL_3_ON) {
          setLED3(ON);
        } else if (output ≡ SL_3_OFF) {
          setLED3(OFF);
        } else if (output ≡ SL_4_ON) {
          setLED4(ON);
        } else if (output ≡ SL_4_OFF) {
          setLED4(OFF);
        } else if (output ≡ SL_5_ON) {
          setLED5(ON);
        } else if (output ≡ SL_5_OFF) {
          setLED5(OFF);
        } else if (output ≡ GO_ON) { // race start
          race.start();
          setGO(ON);
          relayLEDsRed(OFF);
        } else if (output ≡ GO_OFF) { // track call, segment or heat end
          race.pause();
          setGO(OFF);
        } else if (output ≡ STOP_ON) {
          setSTOP(ON);
          if (race.isPaused() ∧ race.fromState(RACE_STARTED)) { // blink
            setLED1(OFF);
            setLED2(ON);
            setLED3(OFF);
            setLED4(ON);
            setLED5(OFF);
            yellowLEDs(ON);
          }
        } else if (output ≡ STOP_OFF) {
          setSTOP(OFF);
          // flickers when race is continued (track or segment)
          if (race.isPaused() ∧
              race.fromState(RACE_STARTED) ∧
              race.areStartingLights(OFF)) { // blink
            setLED1(ON);
            setLED2(OFF);
            setLED3(ON);
            setLED4(OFF);
            setLED5(ON);
            yellowLEDs(OFF);
          }
        } else if (output ≡ PWR_ON) {
          setALL(ON);
          yellowLEDs(ON);
          if (race.isFinished()) {
            setPower(ON);
          }
        } else if (output ≡ PWR_OFF) {
          setALL(OFF);
          if (race.isFinished()) {
            setPower(OFF);
          }
        } 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();
    //  raceStartPauseRestart.isButtonPressed();
    //  powerOff.isButtonPressed();
    //  powerOn.isButtonPressed();
    //  endOfRace.isButtonPressed();
    //  togglePower.isButtonPressed();
    //  toggleYellowFlag.isButtonPressed();
    //  stopAndGoLane1.isButtonPressed();
    //  stopAndGoLane2.isButtonPressed();
    //  stopAndGoLane3.isButtonPressed();
    //  stopAndGoLane4.isButtonPressed();
    //  stopAndGoLane5.isButtonPressed();
    //  stopAndGoLane6.isButtonPressed();
    delay(3);
    attachAllInterrupts();
  }
```