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

   (C) Copyright 2016-2018 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
   ____________ ____________________ ____________________________________________________
   2017-05-20 Gabrile Inäbnit      Slimming down functionality and reduce to four lanes
   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 unsigned long laneProtectionTime = 3000L; // 3 seconds protection time
const byte laneToInterrupMapping[] = { 2, 3, 20, 21 };
const byte laneToRelayMapping[]    = { 31, 32, 33, 34 };
const char lapTime[][7] =
{
  "[SF01$",
  "[SF02$",
  "[SF03$",
  "[SF04$"
};

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

/*****************************************************************************************
   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]"

/*****************************************************************************************
   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 LED_1 26
#define LED_2 25
#define LED_3 24
#define LED_4 23
#define LED_5 22

#define LED_GO 27
#define LED_CAUTION 28
#define LED_STOP 29

// PWR_x: x = lane
#define PWR_ALL 30

// False Start bits
#define FSbit_0 10
#define FSbit_1 8
#define FSbit_2 6
#define FSbit_3 4

// Buttons
#define RACE_START 41
#define RACE_RESTART 42
#define RACE_PAUSE 43
//#define POWER_TOGGLE 46
//#define RACE_START_PAUSE_RESTART 42

/*****************************************************************************************
   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"
```

```
/*****************************************************************************************
  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 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;
    }
```

```
235       bool isInit() {
            return state == RACE_INIT;
          }
          bool fromState(char from) {
            return from == previousState;
240       }
          void init() {
            previousState = state;
            state = RACE_INIT;
          }
245       void start() {
            previousState = state;
            state = RACE_STARTED;
            penaltyStart();
          }
250       void pause() {
            previousState = state;
            state = RACE_PAUSED;
          }
          void finish() {
255         previousState = state;
            state = RACE_FINISHED;
          }
          void startingLightsOn() {
            startingLights = HIGH;
260       }
          void startingLightsOff() {
            startingLights = LOW;
          }
          bool areStartingLightsOff() {
265         return startingLights == LOW;
          }
          bool areStartingLightsOn() {
            return startingLights == HIGH;
          }
270   };

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

      /*************************************************************************************
         Class Lane
       *************************************************************************************/
280   class Lane {
        protected:
          volatile unsigned long start;
          volatile unsigned long finish;
          volatile unsigned long now;
285       volatile long count;
          volatile bool reported;
          byte lane;
          byte pin;
          bool falseStart;
290     public:
          Lane(byte setLane) {
            start = 0L;
            finish = 0L;
            count = -1L;
295         lane = setLane - 1;
            pin = laneToRelayMapping[lane];
            reported = true;
            falseStart = false;
          }
300       void lapDetected() { // called by ISR, short and sweet
            now = millis();
            if ((now - finish) < laneProtectionTime) {
              return;
            }
305         start = finish;
            finish = now;
            count++;
            reported = false;
          }
310       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);
        }
      }
      void powerOff() {
        digitalWrite(pin, LOW);
      }
      bool isFalseStart() {
        return falseStart;
      }
};

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

/*********************************************************************************************
   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,     RACE_START,   10);
Button raceRestart(BUTTON_RACE_RESTART, RACE_RESTART, 10);
Button racePause(BUTTON_RACE_PAUSE,     RACE_PAUSE,   10);

/*****************************************************************************************
   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();
    }
  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
  }
  // interrup pins
  pinMode(LANE_1, INPUT_PULLUP);
  pinMode(LANE_2, INPUT_PULLUP);
  pinMode(LANE_3, INPUT_PULLUP);
  pinMode(LANE_4, 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);
  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 pin;
  public:
    LED (byte setPin) {
      pin = setPin;
      pinMode(pin, OUTPUT);
    }
    void on() {
      digitalWrite(pin, LOW);
    }
    void off() {
      digitalWrite(pin, HIGH);
    }
};

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 ledCaution(LED_CAUTION);
LED ledPowerAll(PWR_ALL);


/*********************************************************************************************
   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);

/*********************************************************************************************
   engage/disengage relays
 *********************************************************************************************/
```

```
void allRelaysOn() {
  relay1.on();
  relay2.on();
  relay3.on();
  relay4.on();
}

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

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

/*****************************************************************************************
   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);
}

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

/*****************************************************************************************
   Interrup Service Routines (ISR) definitions
 *****************************************************************************************/
void lapDetected1() {
  lane1.lapDetected();
}
void lapDetected2() {
```

```
625     lane2.lapDetected();
    }
    void lapDetected3() {
      lane3.lapDetected();
    }
630 void lapDetected4() {
      lane4.lapDetected();
    }

    /*****************************************************************************************
635     Main loop
     *****************************************************************************************/
    void loop() {
      detachAllInterrupts();
      while (Serial.available()) {
640     Serial.readStringUntil('[');
        {
          String output = Serial.readStringUntil(']');
          String raceClockState = output.substring(0, 3); // RC#
          // String raceClockTime = output.substring(4, 8); // HH:MM:SS
645       if (raceClockState ≡ "RC0") { // Race Clock - Race Setup
            if (race.fromState(RACE_FINISHED)) {
              setPowerOff();
              ledSTOP.off();
              ledCaution.on();
650         }
            race.init();
            falseStart.init();
            // } else if (raceClockState == "RC1" && !race.isStarted) { // Race Clock - Race Started
            //   race.start(); // misses the first second
655       } else if (raceClockState ≡ "RC2") { // Race Clock - Race Finished
            ledCaution.off();
            startFinishLED1.on();
            startFinishLED2.on();
            startFinishLED3.on();
660         startFinishLED4.on();
            startFinishLED5.on();
            ledSTOP.on();
            race.finish();
          } else if (raceClockState ≡ "RC3" ∧ ¬race.isPaused()) { // Race Clock - Race Paused
665         race.pause(); // track call immediate, segment end after detection delay
          } else if (output ≡ SL_1_ON) {
            race.startingLightsOn(); // set race starting light state with LED1 only
            startFinishLED1.on();
          } else if (output ≡ SL_1_OFF) {
670         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) {
675         startFinishLED2.off();
          } else if (output ≡ SL_3_ON) {
            startFinishLED3.on();
          } else if (output ≡ SL_3_OFF) {
            startFinishLED3.off();
680       } else if (output ≡ SL_4_ON) {
            startFinishLED4.on();
          } else if (output ≡ SL_4_OFF) {
            startFinishLED4.off();
          } else if (output ≡ SL_5_ON) {
685         startFinishLED5.on();
          } else if (output ≡ SL_5_OFF) {
            startFinishLED5.off();
          } else if (output ≡ GO_ON) { // race start
            race.start();
690         ledGO.on();
            ledCaution.off();
            ledSTOP.off();
          } else if (output ≡ GO_OFF) { // track call, segment or heat end
            race.pause();
695         ledGO.off();
          } else if (output ≡ STOP_ON) {
            if (race.isPaused()) {
              ledCaution.on();
            } else {
700           ledSTOP.on();
            }
            if (race.isPaused() ∧ race.fromState(RACE_STARTED)) { // blink
```

```
            ledCaution.on();
            startFinishLED1.off();
            startFinishLED2.on();
            startFinishLED3.off();
            startFinishLED4.on();
            startFinishLED5.off();
          }
        } 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
            ledCaution.off();
            startFinishLED1.on();
            startFinishLED2.off();
            startFinishLED3.on();
            startFinishLED4.off();
            startFinishLED5.on();
          }
        } else if (output ≡ PWR_ON) {
          ledPowerAll.on();
          if (race.isStarted()) {
            ledCaution.off();
          }
          if (race.isFinished()) {
            setPowerOn();
          }
        } else if (output ≡ PWR_OFF) {
          ledPowerAll.off();
          if (race.isFinished()) {
            setPowerOff();
          }
          if (race.isPaused()) {
            ledCaution.on();
          }
        } 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();
        }
      }
    }
    /** report lap if necessary */
    lane1.reportLap();
    lane2.reportLap();
    lane3.reportLap();
    lane4.reportLap();
    /** any buttons pressed */
    raceStart.isButtonPressed();
    raceRestart.isButtonPressed();
    racePause.isButtonPressed();
    delay(3);
    attachAllInterrupts();
  }
```