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

   (C) Copyright 2016 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.

   Author: Gabriel Inäbnit
   Date  : 2016-10-14

   TODO:
   - aborting start/restart is bogus
   - void startLights(byte pattern): get them patterns figured out
   - use RC3S00:00:00 to establish end of segment state (S/F lights flicker when continue)

   Revision History
   ____________ ____________________ ____________________________________________________
   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
 *****************************************************************************************/

/*****************************************************************************************
   Symbol definitions
 *****************************************************************************************/
#define LANE_1 2
#define LANE_2 3
#define LANE_3 21
#define LANE_4 20
#define LANE_5 19
#define LANE_6 18

#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"

#define LED_1 5
#define LED_2 6
#define LED_3 7
#define LED_4 8
#define LED_5 9
```

```
#define LED_GO 10
#define LED_STOP 11
#define LED_CAUTION 12

#define PWR_ALL 30
#define PWR_1   31
#define PWR_2   32
#define PWR_3   33
#define PWR_4   34
#define PWR_5   35
#define PWR_6   36

#define FS_0 22
#define FS_1 23
#define FS_2 24
#define FS_3 25

/*****************************************************************************************
   Global variables
 *****************************************************************************************/
const unsigned int serialSpeed = 57600;
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
};

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

```
235       }
          bool fromState(char from) {
            return from ≡ previousState;
          }
          void init() {
240         previousState = state;
            state = RACE_INIT;
          }
          void start() {
            previousState = state;
245         state = RACE_STARTED;
            penaltyStart();
          }
          void pause() {
            previousState = state;
250         state = RACE_PAUSED;
          }
          void finish() {
            previousState = state;
            state = RACE_FINISHED;
255       }
      };

      /*****************************************************************************************
         Class Race instantiations
260    *****************************************************************************************/
      Race race;

      /*****************************************************************************************
         Class Lane
265    *****************************************************************************************/
      class Lane {
        protected:
          volatile unsigned long start;
          volatile unsigned long finish;
270       volatile long count;
          volatile bool reported;
          byte lane;
          byte pin;
          bool falseStart;
275     public:
          Lane(byte setLane) {
            start = 0L;
            finish = 0L;
            count = -1L;
280         lane = setLane - 1;
            pin = setLane + 30;
            reported = true;
            falseStart = false;
          }
285       void lapDetected() { // called by ISR, short and sweet
            start = finish;
            finish = millis();
            count++;
            reported = false;
290       }
          void reset() {
            reported = true;
            falseStart = false;
            count = -1L;
295       }
          void reportLap() {
            if (¬reported) {
              Serial.print(lapTime[lane]);
              Serial.print(finish - start);
300           Serial.println(']');
              reported = true;
            }
            if (race.isFalseStartEnabled()) {
              if (race.isInit() ∧ ¬falseStart ∧ (count ≡ 0)) {
305             // false start detected,
                // switching lane off immediately
                powerOff();
                falseStart = true;
                race.setFalseStartDetected(); // burn the race fuse
310           }
              // 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, LOW);
        }
      }
      void powerOff() {
        digitalWrite(pin, HIGH);
      }
      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;
    bool reported;
    bool pressed;
    void reportButton() {
      Serial.println(button);
      reported = true;
    }
  public:
    Button(String setButton, byte setPin) {
      button = setButton;
      pin = setPin;
      reported = false;
      pressed = false;
      pinMode(pin, INPUT_PULLUP);
    }
    void isButtonPressed() {
      pressed = ¬digitalRead(pin);
      if (¬reported ∧ pressed) {
        reportButton();
      }
      reported = pressed;
    }
};

/*************************************************************************************
   Class Button instantiations
 *************************************************************************************/
Button startRace("[BT01]", 44);
Button restartRace("[BT02]", 48);
Button pauseRace("[BT03]", 43);
//Button startPauseRestartRace("[BT04]", 44);
//Button powerOff("[BT05]", 45);
//Button powerOn("[BT06]", 46);
//Button endOfRace("[BT07]", 47);
//Button togglePower("[BT08]", 48);
//Button toggleYelloFlag("[BT09]", 49);
//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(FS_0) |
                  ¬digitalRead(FS_1) << 1 |
                  ¬digitalRead(FS_2) << 2 |
                  ¬digitalRead(FS_3) << 3;
      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(FS_0, INPUT_PULLUP);
  pinMode(FS_1, INPUT_PULLUP);
  pinMode(FS_2, INPUT_PULLUP);
  pinMode(FS_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);
  // turn all LEDs off (HIGH = off)
  digitalWrite(LED_1, HIGH);
  digitalWrite(LED_2, HIGH);
  digitalWrite(LED_3, HIGH);
  digitalWrite(LED_4, HIGH);
  digitalWrite(LED_5, HIGH);
  digitalWrite(LED_GO, HIGH);
  digitalWrite(LED_STOP, HIGH);
  //  digitalWrite(LED_CAUTION, HIGH);
  digitalWrite(PWR_ALL, HIGH);
  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
    //falseStart.init();
    relaysOn(LOW); // switch all power relays on (LOW = on)
    // all defined, ready to read/write from/to serial port
    Serial3.begin(serialSpeed);
    while (¬Serial3) {
      // // wait..
    }
    Serial.begin(serialSpeed);
    while (¬Serial) {
      ; // wait for serial port to connect. Needed for native USB
    }
}

#define CLICK 10

void jiggleRelays() {
  relaysOn(LOW);
  delay(CLICK);
  relaysOn(HIGH);
  delay(222);
  relaysOn(LOW);
  delay(CLICK);
  relaysOn(HIGH);
  delay(111);
  relaysOn(LOW);
  delay(CLICK);
  relaysOn(HIGH);
  delay(111);
  relaysOn(LOW);
  delay(CLICK);
  relaysOn(HIGH);
  delay(222);
  relaysOn(LOW);
  delay(CLICK);
  relaysOn(HIGH);
  delay(444);
  relaysOn(LOW);
  delay(CLICK);
  relaysOn(HIGH);
  delay(222);
  relaysOn(LOW);
  delay(CLICK);
  relaysOn(HIGH);
}

void relaysOn (bool onOff) {
  digitalWrite(PWR_1, onOff);
  digitalWrite(PWR_2, onOff);
  digitalWrite(PWR_3, onOff);
  digitalWrite(PWR_4, onOff);
  digitalWrite(PWR_5, onOff);
  digitalWrite(PWR_6, onOff);
}

#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);
}

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);
}

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);
      if (raceClockState == "RC0") { // Race Clock - Race Setup
        if (race.fromState(RACE_FINISHED)) {
          digitalWrite(LED_1, LOW);
          digitalWrite(LED_2, LOW);
          digitalWrite(LED_3, LOW);
          digitalWrite(LED_4, LOW);
          digitalWrite(LED_5, LOW);
        }
        race.init();
        falseStart.init();
        // } else if (raceClockState == "RC1") { // Race Clock - Race Started
        //   race.start(); // misses the first second
        // } else if () { // Race Clock - Race Finished
        //   race.finish(); // not seen from PC Lap Counter
        // } else if (raceClockState == "RC3") { // Race Clock - Race Paused
        //   race.pause(); // kicks in after detection delay
      } else if (race.isPaused() ∧ output == "RC3R00:00:00") { // Race Clock - Rem. Time 00:00:00
        race.finish();
        digitalWrite(LED_1, LOW);
        digitalWrite(LED_2, LOW);
        digitalWrite(LED_3, LOW);
        digitalWrite(LED_4, LOW);
        digitalWrite(LED_5, LOW);
      } else if (output == SL_1_ON) {
        digitalWrite(LED_1, LOW);
      } else if (output == SL_1_OFF) {
        digitalWrite(LED_1, HIGH);
      } else if (output == SL_2_ON) {
        digitalWrite(LED_2, LOW);
      } else if (output == SL_2_OFF) {
        digitalWrite(LED_2, HIGH);
```

```
625         } else if (output ≡ SL_3_ON) {
              digitalWrite(LED_3, LOW);
            } else if (output ≡ SL_3_OFF) {
              digitalWrite(LED_3, HIGH);
            } else if (output ≡ SL_4_ON) {
630           digitalWrite(LED_4, LOW);
            } else if (output ≡ SL_4_OFF) {
              digitalWrite(LED_4, HIGH);
            } else if (output ≡ SL_5_ON) {
              digitalWrite(LED_5, LOW);
635         } else if (output ≡ SL_5_OFF) {
              digitalWrite(LED_5, HIGH);
            } else if (output ≡ GO_ON) { // race start
              race.start();
              digitalWrite(LED_GO, LOW);
640         } else if (output ≡ GO_OFF) { // track call, segment or heat end
              race.pause();
              digitalWrite(LED_GO, HIGH);
            } else if (output ≡ STOP_ON) {
              digitalWrite(LED_STOP, LOW);
645           if (race.isPaused() ∧ race.fromState(RACE_STARTED)) { // blink
                digitalWrite(LED_1, HIGH);
                digitalWrite(LED_2, LOW);
                digitalWrite(LED_3, HIGH);
                digitalWrite(LED_4, LOW);
650             digitalWrite(LED_5, HIGH);
              }
            } else if (output ≡ STOP_OFF) {
              digitalWrite(LED_STOP, HIGH);
              if (race.isPaused() ∧ race.fromState(RACE_STARTED)) { // blink
655             digitalWrite(LED_1, LOW);
                digitalWrite(LED_2, HIGH);
                digitalWrite(LED_3, LOW);
                digitalWrite(LED_4, HIGH);
                digitalWrite(LED_5, LOW);
660           }
            } else if (output ≡ PWR_ON) {
              digitalWrite(PWR_ALL, LOW);
            } else if (output ≡ PWR_OFF) {
              digitalWrite(PWR_ALL, HIGH);
665         } else if (output ≡ PWR_1_ON) {
              lane1.powerOn();
            } else if (output ≡ PWR_1_OFF) {
              lane1.powerOff();
            } else if (output ≡ PWR_2_ON) {
670           lane2.powerOn();
            } else if (output ≡ PWR_2_OFF) {
              lane2.powerOff();
            } else if (output ≡ PWR_3_ON) {
              lane3.powerOn();
675         } else if (output ≡ PWR_3_OFF) {
              lane3.powerOff();
            } else if (output ≡ PWR_4_ON) {
              lane4.powerOn();
            } else if (output ≡ PWR_4_OFF) {
680           lane4.powerOff();
            } else if (output ≡ PWR_5_ON) {
              lane5.powerOn();
            } else if (output ≡ PWR_5_OFF) {
              lane5.powerOff();
685         } else if (output ≡ PWR_6_ON) {
              lane6.powerOn();
            } else if (output ≡ PWR_6_OFF) {
              lane6.powerOff();
            } else if (raceClockState ≡ "DEV") {
690           race.debug();
            }
          }
        }
        /** report lap if necessary */
695     lane1.reportLap();
        lane2.reportLap();
        lane3.reportLap();
        lane4.reportLap();
        lane5.reportLap();
700     lane6.reportLap();
        /** any buttons pressed */
        startRace.isButtonPressed();
```

```
      restartRace.isButtonPressed();
      pauseRace.isButtonPressed();
      //  startPauseRestartRace.isButtonPressed();
      //  powerOff.isButtonPressed();
      //  powerOn.isButtonPressed();
      //  endOfRace.isButtonPressed();
      //  togglePower.isButtonPressed();
      //  toggleYelloFlag.isButtonPressed();
      //  stopAndGoLane1.isButtonPressed();
      //  stopAndGoLane2.isButtonPressed();
      //  stopAndGoLane3.isButtonPressed();
      //  stopAndGoLane4.isButtonPressed();
      //  stopAndGoLane5.isButtonPressed();
      //  stopAndGoLane6.isButtonPressed();
      delay(3);
      attachAllInterrupts();
    }

```