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

   Revision History
   ________ _____________________ ____________________________________________________
   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_SETUP '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;
    bool falseStartEnabled;
    bool falseStartDetected;
    unsigned long falseStartPenaltyBegin;
    unsigned long falseStartPenaltyServed;
    unsigned long falseStartPenaltyMillis;
  public:
    Race() {
      state = RACE_FINISHED;
      falseStartEnabled = false;
      falseStartDetected = false;
      falseStartPenaltyBegin = 0L;
      falseStartPenaltyServed = 0L;
      falseStartPenaltyMillis = 0L;
    }
    void setFalseStartEnabled(bool yesOrNo, byte penaltyIndex) {
      falseStartEnabled = yesOrNo;
      if (falseStartEnabled) { // false start HW enabled
        falseStartDetected = false;
        falseStartPenaltyBegin = 0xFFFFFFFF;
        falseStartPenaltyServed = 0;
        falseStartPenaltyMillis = delayMillis[penaltyIndex];
      }
    }
    void setFalseStartDetected(bool yesOrNo) {
```

```
155         falseStartDetected = yesOrNo;
          }
          bool isFalseStartPenaltyServed() {
            // race has been started but during penalty a track call occured
            // we have to suspend the penalty for the duration of the track call
160         // let's assume there is only *ONE* track call during the short false start penalty period
            // while in race paused mode, keep pushing the falseStartPenaltyBegin value up
            unsigned long now = millis();
            if (falseStartDetected ∧ (falseStartPenaltyServed ≡ 0) ∧ isPaused()) {
              falseStartPenaltyServed = now - falseStartPenaltyBegin;
165         } else if (falseStartDetected ∧ isPaused()) {
              falseStartPenaltyBegin = now - falseStartPenaltyMillis + falseStartPenaltyServed;
            }
            return (now - falseStartPenaltyBegin) > falseStartPenaltyMillis;
          }
170       bool isFalseStartDetected() {
            return falseStartDetected;
          }
          bool isFalseStartEnabled() {
            return falseStartEnabled;
175       }
          bool isStarted() {
            return state ≡ RACE_STARTED;
          }
          bool isPaused() {
180         return state ≡ RACE_PAUSED;
          }
          bool isFinished () {
            return state ≡ RACE_FINISHED;
          }
185       bool isInit() {
            return state ≡ RACE_SETUP;
          }
          void init() {
            state = RACE_SETUP;
190       }
          void start() {
            state = RACE_STARTED;
            if (falseStartEnabled ∧ ¬falseStartDetected) {
              falseStartPenaltyBegin = millis();
195         }
          }
          void pause() {
            state = RACE_PAUSED;
            if (falseStartDetected) {
200
            }
          }
          void finish() {
            state = RACE_FINISHED;
205       }
      };

      /*****************************************************************************************
         Class Race instantiations
210    *****************************************************************************************/
      Race race;

      /*****************************************************************************************
         Class Lane
215    *****************************************************************************************/
      class Lane {
        protected:
          volatile unsigned long start;
          volatile unsigned long finish;
220       volatile long count;
          volatile bool reported;
          byte lane;
          byte pin;
          bool falseStart;
225     public:
          Lane(byte setLane) {
            start = 0L;
            finish = 0L;
            count = -1L;
230         lane = setLane - 1;
            pin = setLane + 30;
```

```
        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(true);
          }
          // switch power back on after false start penalty served
          if (falseStart ∧ race.isFalseStartPenaltyServed()) {
            falseStart = false; // reset false start "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 ∧ race.isStarted()) {
        reportButton();
      }
      reported = pressed;
    }
};

/*****************************************************************************************
   Class Button instantiations
 *****************************************************************************************/
//Button startRace("[BT01]", 41);
//Button restartRace("[BT02]", 42);
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.setFalseStartEnabled(mode > 7, mode - 8);
      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);
}

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 shortOutput = output.substring(0, 3);
        if (shortOutput == "RC0") {
          falseStart.init();
          race.init();
//      } else if (shortOutput == "RC1") {
//        race.start(); // misses the first second
//      } else if (shortOutput == "RC3") {
//        race.pause(); // kicks in after detection delay
      } else if (output == SL_1_ON) {
        digitalWrite(LED_1, LOW);
```

```
540         } 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) {
545           digitalWrite(LED_2, HIGH);
            } else if (output == SL_3_ON) {
              digitalWrite(LED_3, LOW);
            } else if (output == SL_3_OFF) {
              digitalWrite(LED_3, HIGH);
550         } else if (output == SL_4_ON) {
              digitalWrite(LED_4, LOW);
            } else if (output == SL_4_OFF) {
              digitalWrite(LED_4, HIGH);
            } else if (output == SL_5_ON) {
555           digitalWrite(LED_5, LOW);
            } else if (output == SL_5_OFF) {
              digitalWrite(LED_5, HIGH);
            } else if (output == GO_ON) { // race start
              race.start();
560           digitalWrite(LED_GO, LOW);
            } else if (output == GO_OFF) {
              race.pause();
              digitalWrite(LED_GO, HIGH);
            } else if (output == STOP_ON) {
565           digitalWrite(LED_STOP, LOW);
            } else if (output == STOP_OFF) {
              digitalWrite(LED_STOP, HIGH);
            } else if (output == PWR_ON) {
              digitalWrite(PWR_ALL, LOW);
570         } else if (output == PWR_OFF) {
              digitalWrite(PWR_ALL, HIGH);
            } else if (output == PWR_1_ON) {
              lane1.powerOn();
            } else if (output == PWR_1_OFF) {
575           lane1.powerOff();
            } else if (output == PWR_2_ON) {
              lane2.powerOn();
            } else if (output == PWR_2_OFF) {
              lane2.powerOff();
580         } else if (output == PWR_3_ON) {
              lane3.powerOn();
            } else if (output == PWR_3_OFF) {
              lane3.powerOff();
            } else if (output == PWR_4_ON) {
585           lane4.powerOn();
            } else if (output == PWR_4_OFF) {
              lane4.powerOff();
            } else if (output == PWR_5_ON) {
              lane5.powerOn();
590         } else if (output == PWR_5_OFF) {
              lane5.powerOff();
            } else if (output == PWR_6_ON) {
              lane6.powerOn();
            } else if (output == PWR_6_OFF) {
595           lane6.powerOff();
            }
          }
        }
        /** report lap if necessary */
600     lane1.reportLap();
        lane2.reportLap();
        lane3.reportLap();
        lane4.reportLap();
        lane5.reportLap();
605     lane6.reportLap();
        /** any buttons pressed */
        //  startRace.isButtonPressed();
        //  restartRace.isButtonPressed();
        pauseRace.isButtonPressed();
610     startPauseRestartRace.isButtonPressed();
        //  powerOff.isButtonPressed();
        //  powerOn.isButtonPressed();
        //  endOfRace.isButtonPressed();
        togglePower.isButtonPressed();
615     //  toggleYelloFlag.isButtonPressed();
        //  stopAndGoLane1.isButtonPressed();
```

```
      //  stopAndGoLane2.isButtonPressed();
      //  stopAndGoLane3.isButtonPressed();
      //  stopAndGoLane4.isButtonPressed();
      //  stopAndGoLane5.isButtonPressed();
      //  stopAndGoLane6.isButtonPressed();
      delay(3);
      attachAllInterrupts();
    }
```