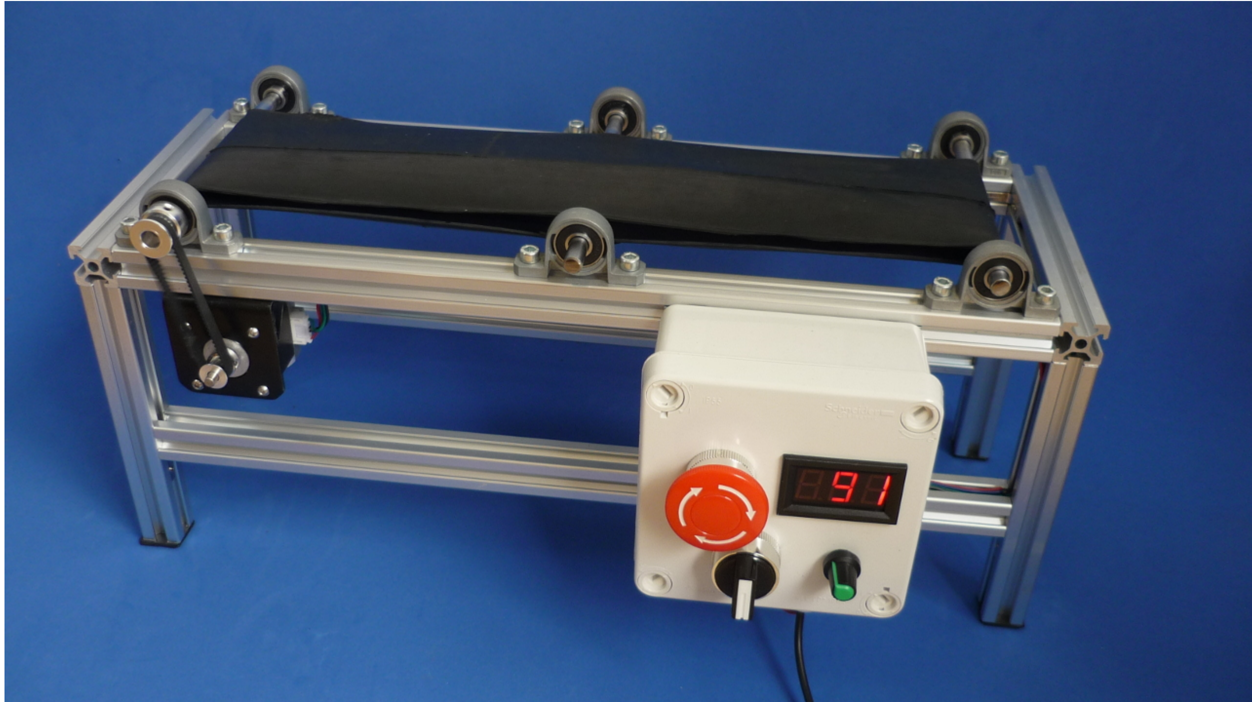
PORTABLE MODULAR CONVEYOR BELT
SPEED CONTROL BY ARDUINO



MODULAR PORTABLE CONVEYOR BELT

The idea of this project is to build a miniature replica of an industrial process, in this particular case a conveyor belt, to be used in educational environments for industrial automation training using PLC, Arduino, Raspberry Pi or any other software programmable platform.

Generally speaking, there are 2 well defined groups of this kind of devices: Ready to use (usually very robust , "good looking") and the "DIY" ones made of household stuff. Ready to use ones are pricey and out of the budget for some schools and institutions, also spare parts only can be obtained from the official manufacturer.

On the other side, the great majority of home made conveyor belts are built with wood and plastics (very often using low cost or recycled material). Those kind of designs are valid as a proof of concept or final course project, but are no strong enough to withstand the daily use in a classroom.

The design shown here tries to get the better of the two worlds:

- Industrial appeal
- Can be assembled and disassembled several times without damage
- Can be stored in small places, easing transportation when not in use.

- Spare (generic brand) parts can be obtained relatively easily and cheap

BASIC FRAME USING INDUSTRIAL COMPONENTS

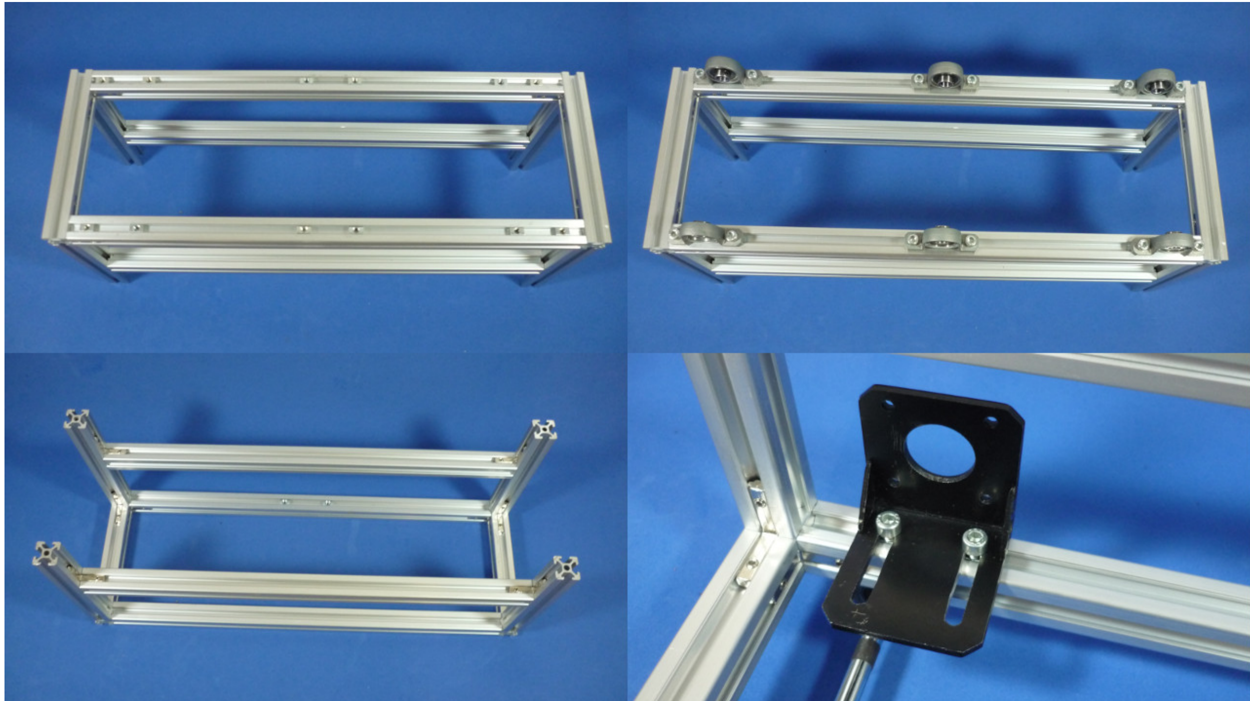
To build the basic structure is, suggested to follow the procedure shown in "Build frame for mini pilot plant" published in an earlier article. In this project the additional 16 (12 for bearings, 2 for motor bracket and 2 for the control box) T slot nuts must be placed inside the slots before building the frame.

The following parts list, is presented as a reference, because is up to you to decide the final geometric dimensions (directly related to the lengths of the chosen profiles) also the amount of axles wanted along the conveyor

- 2020 extruded aluminum T slotted profile "long"
- 2020 extruded aluminum T slotted profile "short"
- Inner T Slot Connector 90 Degree Angle
- 8mm Ball bearing with housing
- M5x8 hex Allen screw
- T-slot nut
- 8mm linear shaft
- GT2 timing belt
- NEMA 17 stepper motor
- NEMA 17 stepper motor bracket
- M3x6 hex Allen screw

- GT2 5 mm 16 teeth pulley
- GT2 8 mm 20 teeth pulley
- 1.5, 2 and 4mm hex Allen keys

.



MOTION TRANSMISSION

The design was conceived to have the least amount of complex mechanical elements as possible, however, two special non trivial steps are required: cut and joint the timing belt that transfers rotation from the motor to the axle, and make the conveyor belt.

<u>Custom length time belt</u>

If there is not a time belt of the desired length in sight, a custom one can be built using a larger one, cutting it and splicing it again to the desired length. The simplest way to accomplish it without specialized products is to sand or remove some teeth for one of the ends of the band, glue overlapped with a rubber glue (don't use super glue or the joint will be too rigid), allow the glue to dry at least 1 hour, then sew it using thread and needle to reinforce the joint. If it's possible, use a timing belt clamp to align properly both ends before gluing.

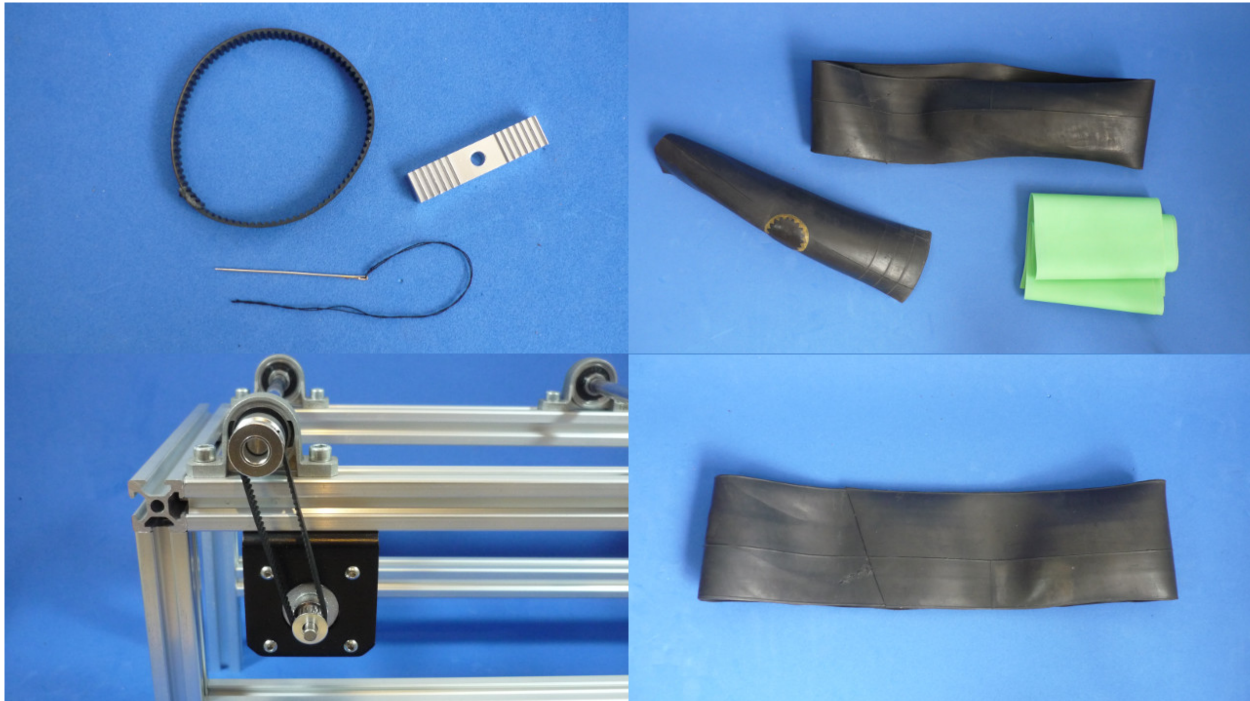
To adjust the tension, move the motor/bracket set along the profile and then secure the screws

<u>Custom made conveyor belt</u>

To avoid the use of additional tensioners, a highly elastic material must be used for the conveyor belt, an old rubber tire tube could be used. If possible from a car, (The bigger , the better) this

provides a more regular and flat surface than smaller ones (i.e. from bikes) . Another alternative, more "good looking" but less robust, could be a rubber resistance band used in yoga and fitness.

The length of the conveyor belt must be estimated to be a little bit shorter than the distance between the farthest axles, so it will auto-tension. Cut the ends of the band in an approximate angle of 45 degrees, so it will have less resistance and smoother travel when the joint approaches to an axle. To join the ends of the band, use a rubber glue (again, don't use super glue or the joint will be too rigid).



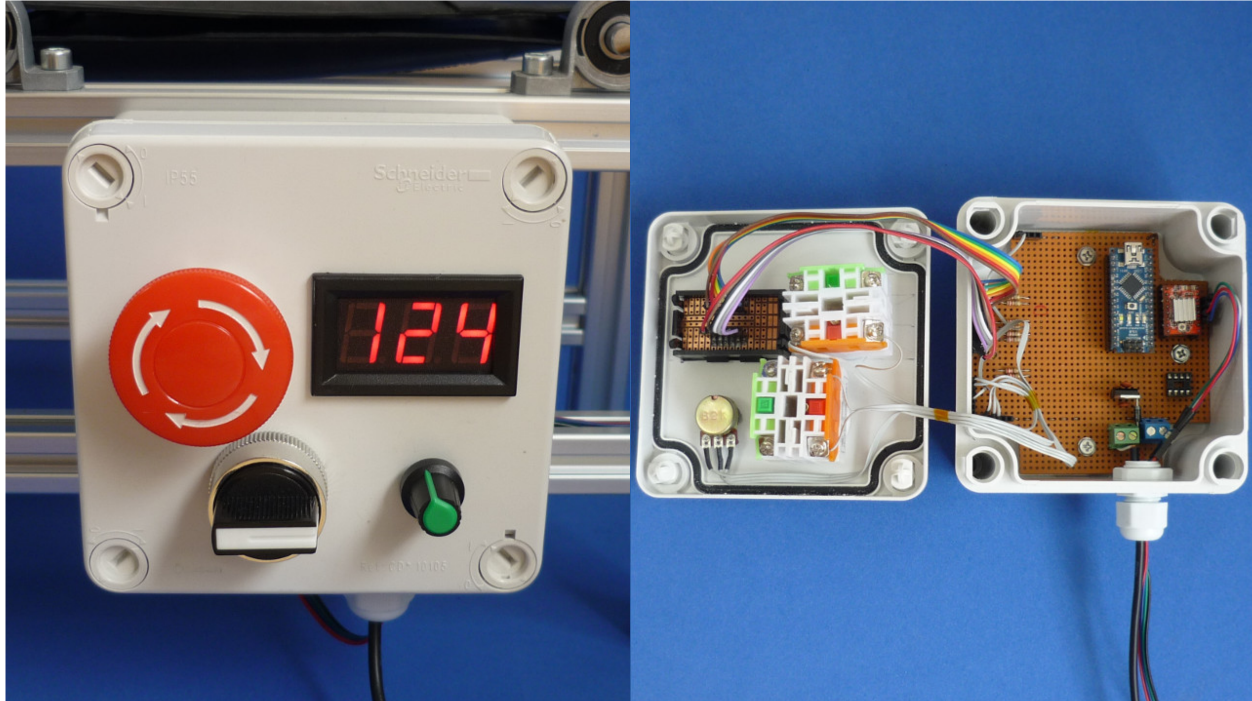
SPEED CONTROL WITH ARDUINO

The speed controller is able to power a stepper motor up to 2 Amps using an A4988 module. An Arduino Nano was used for automating some task, here are:

- 3-digit 7-Segment multiplexed display showing the band's speed in RMP and other messages.
- Emergency stop, once pushed, the only way to release that state is mechanically unlatching the switch and turning control speed to 0.
- Speed control from 0 to 300 RPM approx
- Direction of rotation selector

The circuit was built on a universal PCB and inside a plastic box that can be attached to one of the sides of the conveyor using screws and T slot nuts previously introduced into the slots.

To power the system, a voltage between 14 and 26 V must be applied, and is protected again polarity inversion. A socket for an RS-485 chip was added and wired to RX and TX pins for a future expansion for remote control.



TESTS AND CONCLUSIONS

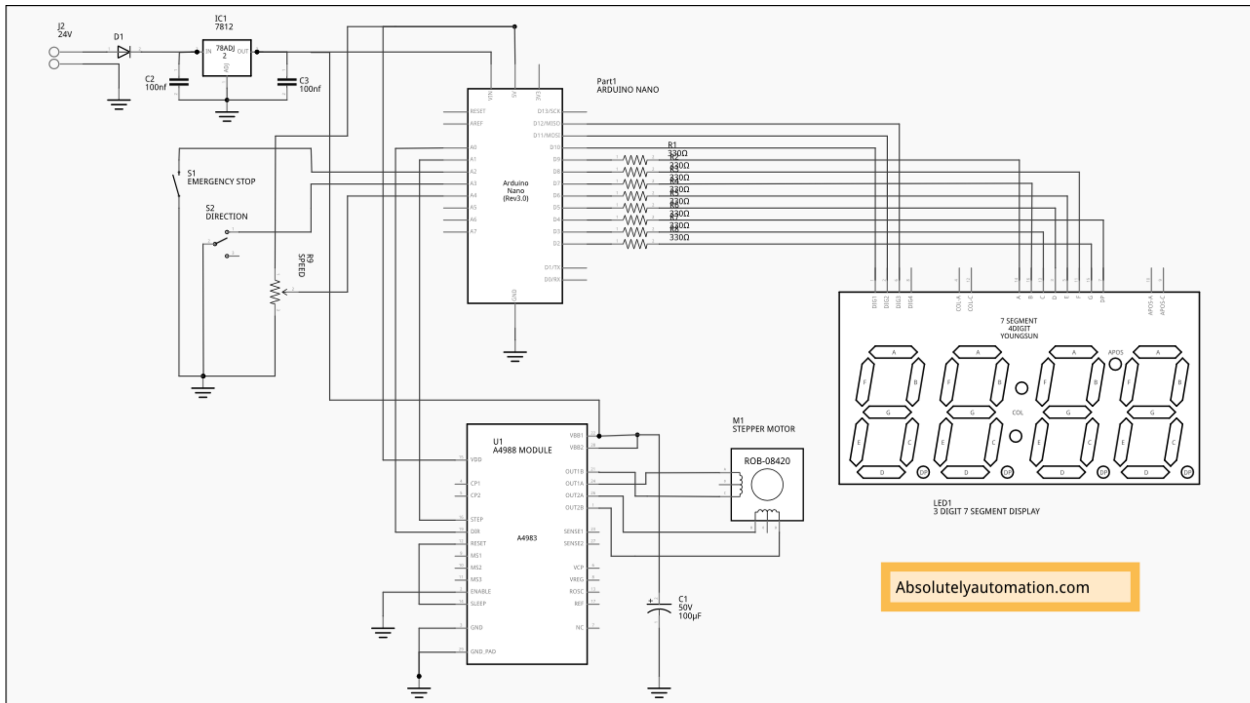
- Plastic bottle caps were used as test subjects on to the band. If you need to move heavier or taller objects, more axles must be added to bring more stability.
- To improve traction, add some "sticky" material to the axle that is moved by the motor. Heat shrink tube is a good candidate
- On very low RPMs considerable vibrations were observed. This phenomena happens to stepper motors when the pulse train frequency is near their natural resonance frequency. Some sort of shock absorbing material must be used between motor and bracket.
- The maximum current of the used motor is 1.2 Amps, so the current limit in the A4988 module was set u to 1 A, because this is the maximum recommended value without using an additional cooling system.
- For a more effective emergency stop, the normally close (N.C) terminal of the switch could be wired in series with the power source of the motors, so if the Arduino fails to stop the motor for some reason, the motion will stop due to lack of electricity.
- To send pulses to the A4988 module the Arduino tone() function was used, so the lowest frequency possible is around 30 Hz.

LINKS

Video that shows all the components and working tests : **youtu.be/X0igjVYxViI**
Arduino Nano Clone: **http://s.click.aliexpress.com/e/FaYvBuf**
A4988 Stepper motor module**: http://s.click.aliexpress.com/e/bqrzjAi**
Rubber resistance band**: http://s.click.aliexpress.com/e/mmqBmma**
2020 extruded aluminum T slotted profile "long":**http://s.click.aliexpress.com/e/EQ7UB2J6a**
2020 extruded aluminum T slotted profile "short":**http://s.click.aliexpress.com/e/fEyFQNN**
Inner T Slot Connector 90 Degree Angle:**http://s.click.aliexpress.com/e/baiqznU**
8mm Ball bearing with housing:**http://s.click.aliexpress.com/e/UZBqrNf**
M5x8 hex Allen screw:**http://s.click.aliexpress.com/e/NV3JieE**
T-slot nut:**http://s.click.aliexpress.com/e/Y3baYJQ**
8mm linear shaft:**http://s.click.aliexpress.com/e/6Y3FQB6**
GT2 timing belt:**http://s.click.aliexpress.com/e/uRfaa2v**
NEMA 17 stepper motor:**http://s.click.aliexpress.com/e/bi6qJun**
NEMA 17 stepper motor bracket:**http://s.click.aliexpress.com/e/iEyZfaq**
M3x6 hex Allen screw:**http://s.click.aliexpress.com/e/vJ6eQf2**
GT2 5 mm 16 teeth pulley:**http://s.click.aliexpress.com/e/R3b6uZz**
GT2 8 mm 20 teeth pulley:**http://s.click.aliexpress.com/e/F62zFQF**
1.5, 2 and 4mm hex Allen keys:**http://s.click.aliexpress.com/e/qFuJuVF**

Tool for opening plastic cases: **http://s.click.aliexpress.com/e/RfAEqnM**
USB to RS-485 converter: **http://s.click.aliexpress.com/e/EUFiuRF**
Clips for PCB DIN rail mount: **http://s.click.aliexpress.com/e/AIIuBIm**

SCHEMATIC FOR THE ARDUINO SPEED CONTROLLED PORTABLE MODULAR CONVEYOR BELT



CONVEYORBELT.INO

```
// ***************************************
//            Absolutelyautomation.com
//
//  Speed control for conveyor belt
//  with stepper motor, A4988 module and 3 digit 7 segments
//  multiplexed display
//  Control panel with emergency stop, direction selector
//  and potentiometer for speed control
//
//****************************************

/*

  3 DIGIT 7 SEGMENT DISPLAY REF 5631AS
  COMMON CATODE

   A

F |‾G| B
   -
E | | C
   - .DP
   D

 */

#include <SevenSeg.h>

// 7 segmentos 3 digitos display pins
int sega = 9;
int segb = 7;
int segc = 3;
int segd = 5;
int sege = 6;
int segf = 8;
int segg = 2;
int segp = 4;

int digit1 = 10;
int digit2 = 11;
int digit3 = 12;

// A4988 stepper control module pins

#define StepperPulse   A1
#define StepperDir     A0
#define StepsRev       200
#define MaxFreq        1000
#define FWD            0
#define REV            1
```

```
// control panel

#define Estop           A2
#define Direction       A3
int Potentiometer = 4;
int ToneFreqMin = 31;    //upper limit for tone() !!

SevenSeg disp( sega, segb, segc, segd, sege, segf, segg );
const int numOfDigits = 3;
int digitPins[numOfDigits] = { digit1, digit2, digit3  };

int firstboot;
int emergencystop;
int directionstop;
int lastdirection;
int potvalue;
int RPM;
unsigned int FREQ;

void setup() {

  // Display configuration

  pinMode(sega, OUTPUT);
  pinMode(segb, OUTPUT);
  pinMode(segc, OUTPUT);
  pinMode(segd, OUTPUT);
  pinMode(sege, OUTPUT);
  pinMode(segf, OUTPUT);
  pinMode(segg, OUTPUT);
  pinMode(segp, OUTPUT);

  pinMode(digit1, OUTPUT);
  pinMode(digit2, OUTPUT);
  pinMode(digit3, OUTPUT);

  disp.setDPPin(segp);
  disp.setCommonCathode();
  disp.setDigitPins( numOfDigits, digitPins );
  disp.setTimer(1);  // Don't use with servo functions at the same time!
  //disp.setTimer(2);  // Don't use with the tone() function at the same time!
  disp.startTimer();

  // Pin setup for the A4988 stepper module

  pinMode(StepperPulse, OUTPUT);
  pinMode(StepperDir, OUTPUT);

  // Pin setup for the control panel
  pinMode(Estop, INPUT);
  pinMode(Direction, INPUT);
  digitalWrite(Estop, HIGH);      // PullUp active
  digitalWrite(Direction, HIGH);  // PullUp active
```

```
}

// infinite loop:
void loop() {
  if(!firstboot){
    digitalWrite(StepperPulse, LOW);
    digitalWrite(StepperDir, FWD);
    lastdirection=REV;
    noTone(StepperPulse);
    disp.write("8.8.8.");
    delay(500);
    disp.write("");
    delay(500);
    disp.write("ON");
    delay(500);
    disp.write("");
    delay(500);
    disp.write("ON");
    delay(500);
    disp.write("");
    delay(500);
    firstboot=1;
  }

  potvalue = analogRead(Potentiometer);
  FREQ = map(potvalue, 0, 1023, 0, MaxFreq);

  // put emergencystop=TRUE when emergency stop pushed
  if( digitalRead(Estop) == 1 ){
    emergencystop = 1;
  }

  // Stop motor and show message
  if(emergencystop == 1){
    noTone(StepperPulse);
    disp.write("Est");
  }


  // To exit from emergency stop state, pull unlatch the switch and set speed
control to 0
  if(digitalRead(Estop) == 0 && emergencystop == 1 &&  FREQ < ToneFreqMin ){
    emergencystop =0;
  }



  // Rotation direction change

  if(digitalRead(Direction) == FWD && lastdirection == REV){
    noTone(StepperPulse);
    digitalWrite(StepperDir, FWD);
    lastdirection = FWD;
```

```
  }

  if(digitalRead(Direction) == REV && lastdirection == FWD){
    noTone(StepperPulse);
    digitalWrite(StepperDir, REV);
    lastdirection = REV;
  }



  // Update display (RPM)
  if(emergencystop != 1 ){

    if( FREQ < ToneFreqMin){
      RPM = 0 ;
      noTone(StepperPulse);
    }
    else
    {
      RPM = (FREQ *60 / StepsRev ) ;
      tone(StepperPulse, FREQ);
    }
    disp.write(RPM);

  }



}

ISR(TIMER1_COMPA_vect)   {
  disp.interruptAction();
}
```

SevenSeg.h

```
/*
  SevenSeg v1.1
  SevenSeg.h - Library for controlling a 7-segment LCD
  Created by Sigvald Marholm, 02.06.2015.
*/

#ifndef SevenSeg_h
#define SevenSeg_h

#include "Arduino.h"

class SevenSeg
{

  public:

    // Constructor
    SevenSeg(int,int,int,int,int,int,int);

    // Low level functions for initializing hardware
    void setCommonAnode();
    void setCommonCathode();
    void setDigitPins(int,int *);
    void setActivePinState(int,int);
    void setDPPin(int);
    void setColonPin(int);
    void setSymbPins(int,int,int,int);

    // Low level functions for printing to display
    void clearDisp();
    void changeDigit(int);
    void changeDigit(char);
    void writeDigit(int);
    void writeDigit(char);
    void setDP();
    void clearDP();
    void setColon();
    void clearColon();
    void setApos();
    void clearApos();

    // Low level functions for controlling multiplexing
    void setDigitDelay(long int);   // Should I have this function?
    void setRefreshRate(int);
    void setDutyCycle(int);

    // High level functions for printing to display
    void write(long int);
    void write(int);
    void write(long int,int);
    void write(int, int);
    void write(char*);
```

```
    void write(String);
    void write(double);
    void write(double num, int point);
    void writeClock(int,int,char);
    void writeClock(int,int);
    void writeClock(int,char);
    void writeClock(int);

    // Timer control functions
    void setTimer(int);
    void clearTimer();
    void interruptAction();
    void startTimer();
    void stopTimer();

    // To clean up
//  void setPinState(int);    // I think this isn't in use. Its called
setActivePinState?
//  int getDigitDelay();      // How many get-functions should I make?

  private:

    // The pins for each of the seven segments (eight with decimal point)
    int _A;
    int _B;
    int _C;
    int _D;
    int _E;
    int _F;
    int _G;
    int _DP;      // -1 when decimal point not assigned

    // Variables used for colon and apostrophe symbols
    int _colonState;    // Whether colon is on (_segOn) or off (_segOff).
    int _aposState;     // Whether apostorphe is on (_segOn) or off (_segOff).
    int _colonSegPin;
    int _colonSegLPin;
    int _aposSegPin;
    int _symbDigPin;

    /* The colon/apostrophe handling needs some further explanation:
     *
     * colonSegPin is the segment pin for colon. I.e. some displays have a
separate segment for colon on one of the digits.
     * others have colon split across two digits: i.e. the upper part has a
separate segment on one digit, whereas the lower
     * part has uses the same segment pin but on another digit. It is assumed
that this segment pin is only used for colon,
     * and it is stored into colonSegPin by setColonPin(int). The functions
setColon() and clearColon() turns on/off this pin,
     * respectively.
     *
     * On some displays, colon is one or two separate free-standing LED(s)
with its own cathode and anode. In case of common
```

```
     * cathode, ground the cathod and treat the anode(s) as a segment pin. The
other way around in case of common anode. This
     * should make the method described above applicable.
     *
     * On other displays, the upper colon part, the lower colon part, as well
as an apostrophe, shares segments with the usual
     * segments (i.e. segments A, B and C) but is treated as a separate symbol
digit that must be multiplexed along with the
     * other digits. In this case the function setSymbPins(int,int,int,int) is
used to assign a pin to that digit, stored in
     * symbDigPin. The pin corresponding to the upper colon segment is stored
in colonSegPin, whereas the lower colon segment
     * is stored in colonSegLPin. aposSegPin holds the segment pin for the
apostrophe. symbDigPin being unequal to -1 is an
     * indication for multiplexing-related functions that it must multiplex
over _numOfDigits+1 digit pins. In this case, the
     * setColon(), clearColon(), setApos() and clearApos() does not directly
influence the pin, but the variable colonState and
     * aposState. In this case, the digit must be changed to the symbol digit
by issuing changeDigit('s') in order to show the
     * symbols.
     */

    // The pins for each of the digits
    int *_dig;
    int _numOfDigits;

    // Timing variables. Stored in microseconds.
    long int _digitDelay;           // How much time spent per display during
multiplexing.
    long int _digitOnDelay;         // How much on-time per display (used for
dimming), i.e. it could be on only 40% of digitDelay
    long int _digitOffDelay;        // digitDelay minus digitOnDelay
    int _dutyCycle;           // The duty cycle (digitOnDelay/digitDelay, here
in percent)
    // Strictly speaking, _digitOnDelay and _digitOffDelay holds redundant
information, but are stored so the computations only
    // needs to be made once. There's an internal update function to update
them based on the _digitDelay and _dutyCycle

    void updDelay();
    void execDelay(int);      // Executes delay in microseconds
    char iaExtractDigit(long int,int,int);
    long int iaLimitInt(long int);

    // Sets which values (HIGH or LOW) pins should have to turn on/off
segments or digits.
    // This depends on whether the display is Common Anode or Common Cathode.
    int _digOn;
    int _digOff;
    int _segOn;
    int _segOff;

    // Variables used by interrupt service routine to keep track of stuff
```

```
    int _timerDigit;          // What digit interrupt timer should update next
time
    int _timerPhase;          // What phase of the cycle it is to update, i.e.
phase 1 (on), or phase 0 (off). Needed for duty cycling.
    int _timerID;       // Values 0,1,2 corresponds to using timer0, timer1 or
timer2.
    long int _timerCounter;         // Prescaler of 64 is used since this is
available on all timers (0, 1 and 2).
                        // Timer registers are not sufficiently large. This
counter variable will extend upon the original timer.
                        // and increment by one each time.
    long int _timerCounterOnEnd;    // How far _timerCounter should count to
provide a delay approximately equal to _digitOnDelay
    long int _timerCounterOffEnd;   // How far _timerCounter should count to
provide a delay approximately equal to _digitOffDelay

    // What is to be printed by interruptAction is determined by these
variables
    long int _writeInt;       // Holds the number to be written in case of
int, fixed point, or clock
    int _writePoint;          // Holds the number of digits to use as decimals
in case of fixed point
//    float _writeFloat;            // Holds the float to write in case of
float. OBSOLETE: Float are converted to fixed point
    char *_writeStr;          // Holds a pointer to a string to write in case
of string
    char _writeMode;          // 'p' for fixed point, 'i' for integer, 'f' for
float, ':'/'.'/'_' for clock with according divisor symbol
    String _writeStrObj;

};

#endif
```

SevenSeg.cpp

```
/*
  SevenSeg v.1.0
  SevenSeg.h - Library for controlling a 7-segment display
  Created by Sigvald Marholm, 02.06.2015.
*/

#include "Arduino.h"
#include "SevenSeg.h"

// Constructor
SevenSeg::SevenSeg(int A,int B,int C,int D,int E,int F,int G){

  // Assume Common Anode (user must change this if false)
  setCommonAnode();

  // Set segment pins
  _A=A;
  _B=B;
  _C=C;
  _D=D;
  _E=E;
  _F=F;
  _G=G;
  _DP=-1;   // DP initially not assigned

  // Set all segment pins as outputs
  pinMode(_A, OUTPUT);
  pinMode(_B, OUTPUT);
  pinMode(_C, OUTPUT);
  pinMode(_D, OUTPUT);
  pinMode(_E, OUTPUT);
  pinMode(_F, OUTPUT);
  pinMode(_G, OUTPUT);

  // Assume no digit pins are used (i.e. it's only one hardwired digit)
  _numOfDigits=0;

  _colonState=_segOff;  // default off
  _aposState=_segOff;   // default off
  _colonSegPin=-1;      // -1 when not assigned
  _colonSegLPin=-1;     // -1 when not assigned
  _aposSegPin=-1; // -1 when not assigned
  _symbDigPin=-1; // -1 when not assigned

  // When no pins are used you need not multiplex the output and the delay is
superfluous
  // TBD: Needed for duty cycle control. Add option to differentiate between
0 and 1 digit pins
  _digitDelay=0;
  _digitOnDelay=0;
  _digitOffDelay=0;
  _dutyCycle=100;
```

```
  // Timer data (default values when no timer is assigned)
  _timerDigit=0;
  _timerPhase=1;
  _timerID=-1;
  _timerCounter=0;
  _timerCounterOnEnd=0;
  _timerCounterOffEnd=0;

  _writeInt=0;
  _writePoint=0;
//  _writeFloat=0;
  _writeStr=0;
  _writeMode=' ';

  // Clear display
  clearDisp();
}

void SevenSeg::setTimer(int timerID){

/*
  Assigns timer0, timer1 or timer2 solely to the task of multiplexing the
display (depending on
  the value of timerNumber).

  For an example of a 5 digit display with 100Hz refresh rate and able to
resolve duty cycle in
  10%-steps a timing of the following resolution is needed:

    1/( 100Hz * 5 digits * 0.1 ) = 200us

  It is sufficient, but the brightness should be adjustable to more than 10
values. Hence a
  resolution of 16us is selected by setting the prescaler to 64 and the
compare register to 3:

    interrupt delay = (64*(3+1))/16MHz = 16us

  The timerCounter variable is of type unsigned int having a maximum value of
65535. Incrementing
  this at each interrupt and taking action upon timerCounterOnEnd or
timerCounterOffEnd yields
  a maximum delay for something to happen:

    max delay for something to happen = 16us * (65535+1) = 1.04s

  which should be more than sufficient if you want to be able to look at your
display.
*/

  _timerID = timerID;

}
```

```
void SevenSeg::clearTimer(){

  stopTimer();
  _timerID = -1;

}

void SevenSeg::startTimer(){

  cli();  // Temporarily stop interrupts

  // See registers in ATmega328 datasheet

  if(_timerID==0){
    TCCR0A = 0;
    TCCR0B = 0;
    TCNT0 = 0;                          // Initialize counter value to 0
    OCR0A = 3;                          // Set Compare Match Register to 3
    TCCR0A |= (1<<WGM01);               // Turn on CTC mode
    TCCR0B |= (1<<CS01) | (1<<CS00);          // Set prescaler to 64
    TIMSK0 |= (1<<OCIE0A);              // Enable timer compare interrupt
  }

  if(_timerID==1){
    TCCR1A = 0;
    TCCR1B = 0;
    TCNT1  = 0;                         // Initialize counter value to 0
    OCR1A = 3;                          // Set compare Match Register to 3
    TCCR1B |= (1 << WGM12);             // Turn on CTC mode
    TCCR1B |= (1 << CS11) | (1 << CS10);  // Set prescaler to 64
    TIMSK1 |= (1 << OCIE1A);            // Enable timer compare interrupt
  }

  if(_timerID==2){
    TCCR2A = 0;
    TCCR2B = 0;
    TCNT2  = 0;                         // Initialize counter value to 0
    OCR2A = 3;                          // Set Compare Match Register to 3
    TCCR2A |= (1 << WGM21);             // Turn on CTC mode
    TCCR2B |= (1 << CS22);              // Set prescaler to 64
    TIMSK2 |= (1 << OCIE2A);            // Enable timer compare interrupt
  }

  sei();  // Continue allowing interrupts

  // update delays to get reasonable values to _timerCounterOn/OffEnd.
  updDelay();
  _timerCounter=0;

}

void SevenSeg::stopTimer(){
  if(_timerID==0){
```

```
    TCCR0B = 0;
  }
  if(_timerID==1){
    TCCR1B = 0;
  }
  if(_timerID==2){
    TCCR2B = 0;
  }
}

void SevenSeg::setCommonAnode(){
  _digOn=HIGH;
  _digOff=LOW;
  _segOn=LOW;
  _segOff=HIGH;
}

void SevenSeg::setCommonCathode(){
  _digOn=LOW;
  _digOff=HIGH;
  _segOn=HIGH;
  _segOff=LOW;
}

void SevenSeg::clearDisp(){

  for(int i=0;i<_numOfDigits;i++){
    digitalWrite(_dig[i], _digOff);
  }
  digitalWrite(_A, _segOff);
  digitalWrite(_B, _segOff);
  digitalWrite(_C, _segOff);
  digitalWrite(_D, _segOff);
  digitalWrite(_E, _segOff);
  digitalWrite(_F, _segOff);
  digitalWrite(_G, _segOff);

  if(_DP!=-1){    // Clear DP too if assigned
    digitalWrite(_DP, _segOff);
  }

  if(_symbDigPin!=-1){
    digitalWrite(_symbDigPin, _digOff);
  }

}

/* OLD METHOD
   This method works, but you have to make a (static) array outside the object
and it
   will only be pointed to by this object. That's not very pretty since the
object doesn't
   actually contain all the information about the display. Further on you rely
on the array
```

```
   being declared in a persistent scope and that the user doesn't change it.
*/
void SevenSeg::setDigitPins(int numOfDigits, int *pDigitPins){
  _dig=pDigitPins;
  _numOfDigits=numOfDigits;

  for(int i=0;i<_numOfDigits;i++){
    pinMode(_dig[i],OUTPUT);
  }

  clearDisp();

  // Set the default refresh rate of 100 Hz. If the user wants another refresh
rate this
  // would have to be set after the setDigitPins function.
  setRefreshRate(100);
}

void SevenSeg::setDigitDelay(long int delay){
  _digitDelay=delay;
  updDelay();
}

void SevenSeg::setDutyCycle(int dc){
  _dutyCycle=dc;
  updDelay();
}

void SevenSeg::setActivePinState(int segActive, int digActive){
  _digOn = digActive;
  _digOff = !digActive;
  _segOn = segActive;
  _segOff = !segActive;
}

void SevenSeg::setRefreshRate(int freq){
  long int period = 1000000L/freq;
  long int digitDelay = period/_numOfDigits;

  if(_symbDigPin!=-1){  // Separate symbol pin in use. One more digit to
multiplex across.
    digitDelay = period/(_numOfDigits+1);
  }

  setDigitDelay(digitDelay);
}

/*
 * HIGH LEVEL WRITE-FUNCTIONS
 *
 * High-level write functions should work in the following way:
 *
 * writeClock(int aa, int bb) - writes a time in the following format aa:bb
(or aa.bb if no colon, or aabb if no dp).
```

```
 * writeClock(int aa, int bb, char c) - where char c specifies decimator ("_"
for no decimator)
 * writeClock(int bb) or writeClock(int bb,char c) - writes in the format
aa:bb (or similar) where aa is deduced from bb.
 * write(int a)
 * write(float a)
 * write(int a, int point)
 * write(char* a)
 *
 * For the timer interrupt mode, the value must simply be stored to a private
variable, and printed by interruptAction. For
 * storing, the following is needed:
 *
 * int _writePoint      - Stores the fixed point in case of writeFixed is
used.
 * int _writeInt  - Stores the number for write(int), writeFixed() and
writeClock() (convert to only one number bb for clock)
 * float _writeFloat    - Stores the float for write(float)
 * char* _writeStr      - Stores a pointer to a string to be written. This
pointer must be maintained outside the class
 * String _writeStrObj  - Stores a string object to be written.
 * char _writeMode      - Describes which writing function/mode is used:
 *                      i - write(int)
 *                      f - write(float)
 *                      p - writeFixed()
 *                      s - write(char* a)
 *                      o - write(String)
 *                      : - writeClock() with colon as decimator
 *                      . - writeClock() with period as decimator
 *                      _ - writeClock() with no decimator
 *
 * Further on, all functions should be using timer if _timerID!=-1. If not,
the should multiplex through the
 * display once and rely on the function being placed in a loop.
 *
 * Perhaps the write int/float functions need a separate private parsing
function to extract the digits? Here's the algorithm:
 *
 *   if num > (10^_numOfDigits-1) or num < (-10^_(numOfDigits-1)+1)     //
Store these values in object during digit pin assignment to save computation?
 *     Display a positive or negative overload
 *
 *   else display can handle number
 *
 *     num = 2468                                    // example
 *     digit_0 = num / (10^(_numOfDigits-1))               // 2
 *     digit_1 = (num / (10^(_numOfDigits-2)))%10          // 4
 *     i_th_digit = (num / (10^(_numOfDigits-1-i)))%10           // 6 and 8
for i=2 and 3
 *
 *     IMPROVED:
 *     num = 2468                              // exampmle
 *     digit_0 = num % 10;
 *     num /= 10;
```

```
 *     digit_1 = num % 10;
 *     num /= 10;
 *
 *    FIXED POINT:
 *    write similar to int, but write fp at correct position.
 *    num=1234, and fp=0 => 1234 (don't show .)
 *    num=1234, and fp=1 => 123.4
 *      num=1234, and fp=4 => 0.123 (option 1, too unpredictable and heavy)
 *    num=1234, and fp=4 => 1234 (simply don't show fp as it's invalid)
 *
 */

void SevenSeg::writeClock(int ss){

  writeClock(ss/60,ss%60);

}

void SevenSeg::writeClock(int ss, char c){

  writeClock(ss/60,ss%60,c);

}

void SevenSeg::writeClock(int mm, int ss){

  // Use ':' if assigned, '.' otherwise, or simply nothing if none assigned

  if(_colonSegPin!=-1){
    writeClock(mm,ss,':');
  } else if(_DP!=-1){
    writeClock(mm,ss,'.');
  } else {
    writeClock(mm,ss,'_');
  }

}

void SevenSeg::writeClock(int mm, int ss, char c){

  if(_timerID==-1){  // No timer assigned. MUX once.

    int num = mm*100+ss;

    // colon through symbpin? 1 if yes.
    int symbColon = (_symbDigPin!=-1);

    for(int i=_numOfDigits-1;i>=0;i--){
      changeDigit(i);
      int nextDigit = num % 10;
      writeDigit(nextDigit);       // Possible future update: don't write
insignificant zeroes
      if(c==':' && !symbColon) setColon();
```

```
      if((c=='.')&&(i==_numOfDigits-3)) setDP();  // Only set "." in the right
place
      num /= 10;
      execDelay(_digitOnDelay);
      if(c==':' && !symbColon) clearColon();
      if(c=='.') clearDP();
      writeDigit(' ');
      execDelay(_digitOffDelay);
    }

    if(symbColon && c==':'){
      changeDigit('s');
      setColon();
      execDelay(_digitOnDelay);
      clearColon();
      execDelay(_digitOffDelay);
    }

  } else {

    _writeMode=c;
    _writeInt=mm*100+ss;

  }

}

void SevenSeg::write(int num,int point){
  write((long int)num, point);
}

void SevenSeg::write(long int num,int point){

  if(_timerID==-1){  // No timer assigned. MUX once.

    // Compute the maximum positive and negative numbers possible to display
    // (TBD: Move this to a computation done on pin assignments?)
    long int maxNegNum=1;
    for(int i=1;i<=_numOfDigits-1;i++) maxNegNum*=10;
    long int maxPosNum=10*maxNegNum-1;
    maxNegNum=-maxNegNum+1;

    // TBD: Change to displaying OL (overload) or ---- or similar?
    if(num>maxPosNum) num=maxPosNum;
    if(num<maxNegNum) num=maxNegNum;

    if(point==0){    // Don't display decimal point if zero decimals used
      point=_numOfDigits;          // value if-sentence won't trigger on
    } else {
      point=_numOfDigits-point-1;  // Map number of decimal points to digit
number
    }

    // TBD: Fix minus
```

```
    int minus=0;
    if(num<0){
      num*=-1;
      minus=1;
    }

/* USED IN v1.0 - DOESN'T SUPPRESS LEADING ZEROS
    for(int i=_numOfDigits-1;i>=0;i--){
      changeDigit(i);
      int nextDigit = num % 10L;
      if(minus&&i==0) writeDigit('-');
      else writeDigit(nextDigit);       // TBD: Possible future update: don't
write insignificant zeroes
      if(point==i) setDP();
      num /= 10;
      execDelay(_digitOnDelay);
      writeDigit(' ');
      clearDP();
      execDelay(_digitOffDelay);
    }
*/

    for(int i=_numOfDigits-1;i>=0;i--){
        changeDigit(i);
        int nextDigit = num % 10L;
        if(num || i>point-1 || i==_numOfDigits-1){
            writeDigit(nextDigit);
        } else if(minus){
            writeDigit('-');
            minus=0;
        } else {
            writeDigit(' ');
        }
        if(point==i) setDP();
        num /= 10;
        execDelay(_digitOnDelay);
        writeDigit(' ');
        clearDP();
        execDelay(_digitOffDelay);
    }

  } else {  // Use timer

    if(point==0){    // Don't display decimal point if zero decimals used
      point=_numOfDigits;          // value if-sentence won't trigger on
    } else {
      point=_numOfDigits-point-1;  // Map number of decimal points to digit
number
    }

      _writeMode = 'p'; // Tell interruptAction that write(int,int) was used
(fixed point).
```

```
      _writeInt = iaLimitInt(num);  // Tell interruptAction to write this
number ...
        _writePoint = point;  // ... with this fixed point
  }

}

// Extracts digit number "digit" from "number" for use with ia -
interruptAction
char SevenSeg::iaExtractDigit(long int number, int digit, int point){

/* OLD VERSION WITHOU ZERO SUPPRESION (v1.0)
  if(number<0){
    if(digit==0) return '-';
    number*=-1;
  }
  for(int i=0;i<_numOfDigits-digit-1;i++) number/=10L;
  return (char)((number%10L)+48L);
*/

  long int old_number = number;
  int minus = 0;
  if(number<0){
    number*=-1;
    minus = 1;
  }

  if(digit!=_numOfDigits-1){
    for(int i=0;i<_numOfDigits-digit-1;i++) number/=10L;
  }

  if(digit>point-1 || digit==_numOfDigits-1 || number!=0) return (char)
((number%10L)+48L);
  else {
    if(iaExtractDigit(old_number,digit+1,point)!='-' && iaExtractDigit
(old_number,digit+1,point)!=' ' && minus) return '-';
    else return ' ';
  }

//  else if(iaExtractDigit(old_number,digit+1,point)=='-' && minus) return
'-';
//  else if(iaExtractDigit(old_number,digit+1,point)!=' ' && iaExtractDigit
(old_number,digit+1,point)!='-' && minus) return '-';
//  else ' ';
/*
  else if(iaExtractDigit(number,digit+1,point)=='0'){
    if(minus) return '-';
    else return ' ';
  } else return ' ';
  */

}

// Limits integer similar to how it's done in write(int,int)
```

```
long int SevenSeg::iaLimitInt(long int number){


    // Compute the maximum positive and negative numbers possible to display
    // (TBD: Move this to a computation done on pin assignments?)
    long int maxNegNum=1;
    for(int i=1;i<=_numOfDigits-1;i++) maxNegNum*=10;
    long int maxPosNum=10*maxNegNum-1;
    maxNegNum=-maxNegNum+1;

    // TBD: Change to displaying OL (overload) or ---- or similar?
    if(number>maxPosNum) number=maxPosNum;
    if(number<maxNegNum) number=maxNegNum;

    return number;

}

void SevenSeg::write(int num){
  write((long int)num);
}

void SevenSeg::write(long int num){

  if(_timerID==-1){  // No timer assigned. MUX once.

    write(num,0);

  } else {  // Use timer

      _writeMode = 'i'; // Tell interruptAction that write(int) is used.
      _writeInt = iaLimitInt(num);  // Tell interruptAction to write this int
  }

}
/*
void SevenSeg::writeDisplay(int A,int B,int C,int D,int colon){

  // Rewrite. Take caution to properly shut down all symbols for the duty
cycle control.
  // This can be done by turning off the digit rather than the symbol, but
that does not
  // work for symbols not connected to any digit (i.e. hardwired). Hence it
should happen
  // at a segment level.x

  int digits[4];
  digits[0]=A;
  digits[1]=B;
  digits[2]=C;
  digits[3]=D;
  for(int i=0;i<_numOfDigits;i++){
```

```
    if(_digitOnDelay!=0){           // delayMicroseconds(0) yields a large
delay for some reason. Hence the if-sentence.
      changeDigit(i);
      writeDigit(digits[i]);
      //if(colon) setColon();
      delayMicroseconds(_digitOnDelay);   // TBD: change to execDelay()
    }
    if(_digitOffDelay!=0){          // delayMicroseconds(0) yields a large
delay for some reason. Hence the if-sentence.
      changeDigit(' ');
      //writeDigit(' ');
      //clearColon();
      delayMicroseconds(_digitOffDelay);// TBD: change to execDelay()
    }
  }
}
*/
void SevenSeg::write(char *str){

  if(_timerID==-1){  // No timer assigned. MUX once.

    int i=0;
    int j=0;
    clearColon();
    while(str[i]!='\0'){
      changeDigit(j);
      writeDigit(str[i]);
      if(str[i+1]=='.'){
        setDP();
        i++;
      }
      execDelay(_digitOnDelay);
      writeDigit(' ');
      clearDP();
      execDelay(_digitOffDelay);
      i++;
      j++;
    }

  } else {  // Use timer
      _writeMode = 's'; // Tell interruptAction that write(char*) is used.
      _writeStr = str;  // Tell interruptAction to write this string
  }


}

void SevenSeg::write(String str){

  if(_timerID==-1){  // No timer assigned. MUX once.

    int i=0;
    int j=0;
    clearColon();
```

```
    while(i<str.length()){
      changeDigit(j);
      writeDigit(str[i]);
      if(str[i+1]=='.'){
        setDP();
        i++;
      }
      execDelay(_digitOnDelay);
      writeDigit(' ');
      clearDP();
      execDelay(_digitOffDelay);
      i++;
      j++;
    }

  } else {  // Use timer

      _writeMode = 'o'; // Tell interruptAction that write(String) is used.
      _writeStrObj = str;     // Tell interruptAction to write this string
  }


}

void SevenSeg::write(double num, int point){
    for(int i=0;i<point;i++) num*=10;
    long int intNum = (long int) num;
    double remainder = num - intNum;
    if(remainder>=0.5 && num>0) intNum++;
    if(remainder<=0.5 && num<0) intNum--;
    write(intNum,point);
}

void SevenSeg::write(double num){

    // Compute the maximum positive and negative numbers possible to display
    // (TBD: Move this to a computation done on pin assignments?)
    long int maxNegNum=1;
    for(int i=1;i<=_numOfDigits-1;i++) maxNegNum*=10;
    long int maxPosNum=10*maxNegNum-1;
    maxNegNum=-maxNegNum+1;

    if(num>maxPosNum) num=maxPosNum;
    if(num<maxNegNum) num=maxNegNum;

    int point=0;
    if(num<0&&num>-1){
      while(num*100<=maxPosNum && num*100>=maxNegNum && point<_numOfDigits-2){
        num*=10;
        point++;
      }
      if((int)num==0){
        point++; // The minus sign will disappear
      }
```

```
    } else if(num>0&&num<1){
      while(num*100<=maxPosNum && num*100>=maxNegNum && point<_numOfDigits-1){
        num*=10;
        point++;
      }
    } else {
      while(num*10<=maxPosNum && num*10>=maxNegNum){
        num*=10;
        point++;
      }
    }

    // Implementing correct round-off
    double rest=num;
    if(rest<0) rest*=-1;
    rest=rest-(long int)rest;
    if(rest>=0.5&&num>0) num++;
    if(rest>=0.5&&num<0) num--;

    if(_timerID==-1){

      write((long int)num,point);

    } else { // user timer

      // Adapting to another format
      point=point+1-_numOfDigits;

      _writeMode='f';
      _writePoint=-point;
      _writeInt=(long int)num;

    }
}

void SevenSeg::updDelay(){

  // On-time for each display is total time spent per digit times the duty
cycle. The
  // off-time is the rest of the cycle for the given display.

  long int temp = _digitDelay;            // Stored into long int since
temporary variable gets larger than 32767
  temp *= _dutyCycle;               // Multiplication in this way to prevent
multiplying two "shorter" ints.
  temp /= 100;                      // Division after multiplication to
minimize round-off errors.
  _digitOnDelay=temp;
  _digitOffDelay=_digitDelay-_digitOnDelay;

  if(_timerID!=-1){
    // Artefacts in duty cycle control appeared when these values changed
while interrupts happening (A kind of stepping in brightness appeared)
    cli();
```

```
    _timerCounterOnEnd=(_digitOnDelay/16)-1;
    _timerCounterOffEnd=(_digitOffDelay/16)-1;
    if(_digitOnDelay==0) _timerCounterOnEnd=0;
    if(_digitOffDelay==0) _timerCounterOffEnd=0;
    _timerCounter=0;
    sei();
  }
}

void SevenSeg::interruptAction(){

  // Increment the library's counter
  _timerCounter++;

  // Finished with on-part. Turn off digit, and switch to the off-phase
(_timerPhase=0)
  if((_timerCounter>=_timerCounterOnEnd)&&(_timerPhase==1)){
    _timerCounter=0;
    _timerPhase=0;

    writeDigit(' ');

    // If a write mode using . is used it is reasonable to assume that DP
exists. Clear it (eventhough it might not be on this digit).
    if(_writeMode=='p'||_writeMode=='.'||_writeMode=='s'||_writeMode=='f')
clearDP();
    if(_writeMode==':') clearColon();

  }

  // Finished with the off-part. Switch to next digit and turn it on.
  if((_timerCounter>=_timerCounterOffEnd)&&(_timerPhase==0)){
    _timerCounter=0;
    _timerPhase=1;

    _timerDigit++;

    //if(_timerDigit>=_numOfDigits) _timerDigit=0;
    if(_timerDigit>=_numOfDigits){
      if(_symbDigPin!=-1 && _timerDigit==_numOfDigits){  // Symbol pin in use.
Let _timerDigit=_numOfDigits be used for symbol mux.
      } else { // Finished muxing symbol digit, or symbol pin not in use
        _timerDigit=0;
      }
    }

    if(_timerDigit==_numOfDigits) changeDigit('s');
    else changeDigit(_timerDigit);
//    writeDigit(_timerDigit);

    if(_writeMode=='p'||_writeMode=='f'){ // Fixed point writing (or float)
      writeDigit(iaExtractDigit(_writeInt,_timerDigit,_writePoint));
      if(_writePoint==_timerDigit) setDP();
    }
```

```
    if(_writeMode=='i'){      // Fixed point writing
      writeDigit(iaExtractDigit(_writeInt,_timerDigit,_numOfDigits));
    }

    if(_writeMode==':'||_writeMode=='.'||_writeMode=='_'){

      // colon through symbpin? 1 if yes.
      int symbColon = (_symbDigPin!=-1);

      if(_timerDigit==_numOfDigits){      // Symbol digit
        setColon();
      } else {
        writeDigit(iaExtractDigit(_writeInt,_timerDigit,_numOfDigits));
        if(_writeMode==':' && !symbColon) setColon();
        if((_writeMode=='.')&&(_timerDigit==_numOfDigits-3)) setDP();  // Only
set "." in the right place
      }

    }

    if(_writeMode=='s'){

      // This algorithm must count to the correct letter i in _writeStr for
digit j, since the two may be unmatched
      // and it is impossible to know which letter to write without counting
      int i=0; // which digit
      int j=0; // which digit have it counted to
      while(_writeStr[i]!='\0' && j<_timerDigit){
        if(_writeStr[i+1]=='.'){
          i++;
        }
        i++;
        j++;
      }
      writeDigit(_writeStr[i]);
      if(_writeStr[i+1]=='.') setDP();

    }

    if(_writeMode=='o'){

      // This algorithm must count to the correct letter i in _writeStr for
digit j, since the two may be unmatched
      // and it is impossible to know which letter to write without counting
      int i=0; // which digit
      int j=0; // which digit have it counted to
      while(i<_writeStrObj.length() && j<_timerDigit){
        if(_writeStrObj[i+1]=='.'){
          i++;
        }
        i++;
        j++;
      }
```

```
      writeDigit(_writeStrObj[i]);
      if(_writeStrObj[i+1]=='.') setDP();

    }

  }


/*
  // If we're in the on-part of the cycle and has counted as many interrupts
corresponding to one on-phase
  if((_timerCounter>=_timerCounterOnEnd) && (_timerPhase==1)){
    _timerCounter=0;    // Reset the library's counter
    _timerPhase=0;      // Switch to off-phase

    // Turn off this digit
    writeDigit(' ');

  }

  // Similar for the off-phase.
  if((_timerCounter>=_timerCounterOffEnd) && (_timerPhase==0)){
    _timerCounter=0;
    _timerPhase=1;

    // Turn on the next digit
    _timerDigit++;
    if(_timerDigit>=_numOfDigits){
      _timerDigit=0;
    }
    changeDigit(_timerDigit);
    writeDigit(_timerDigit);

  }
*/
}

/*
void SevenSeg::setDigitPins(int numOfDigits, int *pDigitPins){

  if(_numOfDigits>0){
//    delete [] _dig;
    free(_dig);
  }

  _numOfDigits = numOfDigits;

//  _dig = new int[numOfDigits];
  _dig = (int*)malloc(_numOfDigits * sizeof(int));

//  memcpy(_dig, pDigitPins, numOfDigits);
  for(int i=0;i<_numOfDigits;i++){
    _dig[i]=pDigitPins[i];
  }
```

```
}
*/
void SevenSeg::changeDigit(int digit){


  // Turn off all digits/segments first.
  // If you swith on a new digit before turning off the segments you will get
  // a slight shine of the "old" number in the "new" digit.
  clearDisp();
  digitalWrite(_dig[digit], _digOn);

}

void SevenSeg::changeDigit(char digit){

  if(digit=='s'){
    // change to the symbol digit
    clearDisp();
    digitalWrite(_symbDigPin, _digOn);
    digitalWrite(_colonSegPin, _colonState);
    digitalWrite(_colonSegLPin, _colonState);
    digitalWrite(_aposSegPin, _aposState);
  }

  if(digit=' '){
    clearDisp();
  }

}

void SevenSeg::setDPPin(int DPPin){

  _DP=DPPin;
  pinMode(_DP, OUTPUT);

}

void SevenSeg::setDP(){

  digitalWrite(_DP, _segOn);

}

void SevenSeg::clearDP(){

  digitalWrite(_DP, _segOff);

}
/*
    void setDPPin(int);
    void setDP();
    void clearDP();
```

Characters: Colon, apostrophe, comma(DP), randomly assignable symbols?
Most symbols can be confined to one character, but colon should be able to assign in two parts (UC and LC).
Yet two colons are also present on some displays. And on some displays, colons and apostorphes are treated as a separate digit using an additional common cathode/anode.

I've decided to treat the symbols in the following way

DP is assigned as an eight segment for each digit, as this is the only way I've seen it done. Simple.
I want the functions setDPPin(int), setDP(), clearDP(). DP should be cleared at each changeDigit (i.e. in ClearDisp?)
In the parsing function it should be possible to write "1.2.3.4.".

Colon are treated in many different ways on many different displays. I want a function setColonPin() that are overloaded
and take most of the case. setColon() and clearColon() should turn it on or off. I'll explain the scenarios, and the syntax:

Colon may be split in two parts UC (upper colon) and LC (lower colon) or colon may be hardwired as one LED

1. Colon has its own cathodes and anodes. Ground the cathode if common cathode or tie anode to supply in case of common anode.
   Syntax: setColonPin(segPin) where segPin is the other pin. In this case the colon needs not be multiplexed. If split segment pin
   for UC/LC, the user joins them together. setColon()/clearColon() writes directly to segmentPin.
2. UC/LC is joined together using its own segment pin, and shares common anode/cathode with one of the digits.
   Syntax: setColonPin(segPin,digPin). The function will detect at what digit to type the colon based on digitPin, and store this in colonDigU and
   colonDigL. If no digit is tied to digitPin issue an error. colonState is a private member variable being set or cleared by setColon()/clearColon().
   writeDigit() checks colonState when on colonDigU or colonDigL digits and writes accordingly.
3. UC shares common anode/cathode digit pin with one of the digits, while LC shares with another digit. They are the same segment pin.
   Syntax: setColonPin(segPin,digLPin,digUPin). This works in the same way as above, except that different values are stored to DigU and DigL.
4. UC and LC are treated like seperate segments on a new "symbol" digit pin(!). In these cases there is usually also an apostrophe (A) segment.
   The UC and LC segments should be joined together into one segment (C).
   Syntax: setSymbolPin(SegCPin,SegAPin,digPin) digPin will be stored to symbDigPin, and the multiplexing must occur over one additional digit.
   This implies modification to i.e. setRefreshRate, setDigitDelay and maybe other functions. All functions must be checked.

Members needed in class:

```
      private:
        colonState      ;     // _segOn or _segOff.
      aposState;  // _segOn or _segOff.
```

```
        colonDigU;      // Which digits to activate colon at (one digit for UC
and one for LC)
      colonDigL;  // colonDigU==colonDigL==-1 means that it is treated as a
separate digit, case 1 or 4. (check negative numbers for int)
      symbDigPin; // If the colon is on a separate digit pin the symbol digit
pin number is stored here. Otherwise, it is -1. Non-zero values imply more
muxing.
        colonSegPin;
      aposSegPin;
      public:
      setColonPin(int);
        setColonPin(int,int);
      setColonPin(int,int,int);
        setSymbPin(int,int,int,int);
        setColon();
        clearColon();
      setApos();
      clearApos();

  The parsing function should parse colon to off except when any colon present
in string. Same with apostorphe. I.e. "34:07". I initially wanted to have
support for two
  colons since you need that on a watch. However, I've settled on only one
colon since there are almost none display available with two colons. If I find
one, and will
  use one, I will simply duplicate the colon stuff in my class.

  Actually, cases 1, 2 and 3 can be joined together! The colon can be turned
on irrespective of what digits they are on (at least as long as there are only
one colon).
  This simplifies the class:

      private:
        colonState;     // _segOn or _segOff.
      aposState;  // _segOn or _segOff.
      symbDigPin; // If the colon is on a separate digit pin the symbol digit
pin number is stored here. Otherwise, it is -1. Non-zero values imply more
muxing.
        colonSegPin;
      aposSegPin;
      public:
      setColonPin(int);
        setSymbPin(int,int,int,int);
        setColon();
        clearColon();
      setApos();
      clearApos();

*/

void SevenSeg::setColonPin(int colonPin){
  _colonSegPin=colonPin;
  pinMode(_colonSegPin,OUTPUT);
  digitalWrite(_colonSegPin, _colonState);
```

```
}

void SevenSeg::setSymbPins(int digPin, int segUCPin, int segLCPin, int
segAPin){
  _colonSegPin=segUCPin;
  _colonSegLPin=segLCPin;
  _aposSegPin=segAPin;
  _symbDigPin=digPin;
  pinMode(_colonSegPin,OUTPUT);
  pinMode(_colonSegLPin,OUTPUT);
  pinMode(_aposSegPin,OUTPUT);
  pinMode(_symbDigPin,OUTPUT);
  digitalWrite(_colonSegPin, _colonState);
  digitalWrite(_colonSegLPin, _colonState);
  digitalWrite(_aposSegPin, _aposState);
}

/*
The functions for setColon(), clearColon(), setApos(), clearApos() directly
sets or clears the
segment pins if no symbol pin is assigned. Since no symbol pin is assigned
colon (apos isn't set) has
a separate segment pin "Colon" and shares a digit pin with one or two other
digits
(in case it is split into UC and LC). In this case it makes sense to control
it just like other
segments; by setting and clearing the segment pin with setColon() or
clearColon() after the correct
digit is selected with changeDigit(). Compare with setDP()/clearDP().
Furthermore, it is not necessary
to identify WHICH digit the colon segments apply for since, if colon is turned
on, one may simply switch
on the segment pin for all digits. Nothing will be tied to the colon segment
pin for other digits than
those it applies to, hence it is sufficient to initialize this kind of
hardware with setColon(int colonPin).
Sometimes, a colon is present as one or two complete stand-alone LEDs. In this
case, the can be wired
up into one of these configurations to work.

In the other main case, a separate symbol pin is assigned for colon and
apostrophe. This is actually an
additional digit pin which must be muxed across. The segment pins are shared
with other segments such as
A-G. This is a compact way of allowing many symbols (colon and apostrophe)
while only adding one more pin.
This configuration is programmed with setSymbPins(int digPin, int segUCPin,
int segLCPin, int segAPin),
where digPin is the symbol digit pin and the other pins are the pins used for
segment UC, LC and apos.
If colon is present as one segment only, segUCPin and segULPin can be the same
value. Sometimes, colon
and apostorphe are present as stand-alone diodes with their own cathodes and
anodes not being connected
```

```
to anything else. In this case, join their cathodes or anodes (in case of
common cathode or anode respectively)
and connect their other terminal to one segment pin each to make the mentioned
configuration.
The behaviour set/clear behaviour of these digits are a bit different in this
case. The set/clear-funciton
only sets a flag to on or off. In order to type the characters you must mux to
the symbol digit by issuing
changeDigit('s'). This function will light up the appropriate symbols in
accordance with the flags.
*/

void SevenSeg::setColon(){
  _colonState=_segOn;
  if(_symbDigPin==-1){
    digitalWrite(_colonSegPin, _segOn);
  }
}

void SevenSeg::clearColon(){
  _colonState=_segOff;
  if(_symbDigPin==-1){
    digitalWrite(_colonSegPin, _segOff);
  }
}

void SevenSeg::setApos(){
  _aposState=_segOn;
  if(_symbDigPin==-1){
    digitalWrite(_aposSegPin, _segOn);
  }
}

void SevenSeg::clearApos(){
  _aposState=_segOff;
  if(_symbDigPin==-1){
    digitalWrite(_aposSegPin, _segOff);
  }
}

void SevenSeg::writeDigit(int digit){

  // Turn off all LEDs first to avoid running current through too many LEDs at
once.
  digitalWrite(_A, _segOff);
  digitalWrite(_B, _segOff);
  digitalWrite(_C, _segOff);
  digitalWrite(_D, _segOff);
  digitalWrite(_E, _segOff);
  digitalWrite(_F, _segOff);
  digitalWrite(_G, _segOff);

  if(digit==1){
    digitalWrite(_B, _segOn);
```

```
    digitalWrite(_C, _segOn);
  }

  if(digit==2){
    digitalWrite(_A, _segOn);
    digitalWrite(_B, _segOn);
    digitalWrite(_G, _segOn);
    digitalWrite(_E, _segOn);
    digitalWrite(_D, _segOn);
  }

  if(digit==3){
    digitalWrite(_A, _segOn);
    digitalWrite(_B, _segOn);
    digitalWrite(_G, _segOn);
    digitalWrite(_C, _segOn);
    digitalWrite(_D, _segOn);
  }

  if(digit==4){
    digitalWrite(_F, _segOn);
    digitalWrite(_G, _segOn);
    digitalWrite(_B, _segOn);
    digitalWrite(_C, _segOn);
  }

  if(digit==5){
    digitalWrite(_A, _segOn);
    digitalWrite(_F, _segOn);
    digitalWrite(_G, _segOn);
    digitalWrite(_C, _segOn);
    digitalWrite(_D, _segOn);
  }

  if(digit==6){
    digitalWrite(_A, _segOn);
    digitalWrite(_F, _segOn);
    digitalWrite(_E, _segOn);
    digitalWrite(_D, _segOn);
    digitalWrite(_C, _segOn);
    digitalWrite(_G, _segOn);
  }

  if(digit==7){
    digitalWrite(_A, _segOn);
    digitalWrite(_B, _segOn);
    digitalWrite(_C, _segOn);
  }

  if(digit==8){
    digitalWrite(_A, _segOn);
    digitalWrite(_B, _segOn);
    digitalWrite(_C, _segOn);
    digitalWrite(_D, _segOn);
```

```
    digitalWrite(_E, _segOn);
    digitalWrite(_F, _segOn);
    digitalWrite(_G, _segOn);
  }

  if(digit==9){
    digitalWrite(_G, _segOn);
    digitalWrite(_F, _segOn);
    digitalWrite(_A, _segOn);
    digitalWrite(_B, _segOn);
    digitalWrite(_C, _segOn);
    digitalWrite(_D, _segOn);
  }

  if(digit==0){
    digitalWrite(_A, _segOn);
    digitalWrite(_B, _segOn);
    digitalWrite(_C, _segOn);
    digitalWrite(_D, _segOn);
    digitalWrite(_E, _segOn);
    digitalWrite(_F, _segOn);
  }

}

void SevenSeg::writeDigit(char digit){

  // Turn off all LEDs first. Run writeDigit(' ') to clear digit.
  digitalWrite(_A, _segOff);
  digitalWrite(_B, _segOff);
  digitalWrite(_C, _segOff);
  digitalWrite(_D, _segOff);
  digitalWrite(_E, _segOff);
  digitalWrite(_F, _segOff);
  digitalWrite(_G, _segOff);

  if(digit=='-'){
    digitalWrite(_G, _segOn);
  }

  if(digit=='\370'){ // ASCII code 248 or degree symbol: '°'
    digitalWrite(_A, _segOn);
    digitalWrite(_B, _segOn);
    digitalWrite(_F, _segOn);
    digitalWrite(_G, _segOn);
  }

  // Digits are numbers. Write with writeDigit(int)
  if(digit>=48&&digit<=57)  writeDigit(digit-48);

  // Digits are small caps letters. Capitalize.
  if(digit>=97&&digit<=122) digit-=32;

  if(digit=='A'){
```

```
    digitalWrite(_A, _segOn);
    digitalWrite(_B, _segOn);
    digitalWrite(_C, _segOn);
    digitalWrite(_E, _segOn);
    digitalWrite(_F, _segOn);
    digitalWrite(_G, _segOn);
  }

  if(digit=='B'){
    digitalWrite(_C, _segOn);
    digitalWrite(_D, _segOn);
    digitalWrite(_E, _segOn);
    digitalWrite(_F, _segOn);
    digitalWrite(_G, _segOn);
  }

  if(digit=='C'){
    digitalWrite(_A, _segOn);
    digitalWrite(_D, _segOn);
    digitalWrite(_E, _segOn);
    digitalWrite(_F, _segOn);
  }

  if(digit=='D'){
    digitalWrite(_B, _segOn);
    digitalWrite(_C, _segOn);
    digitalWrite(_D, _segOn);
    digitalWrite(_E, _segOn);
    digitalWrite(_G, _segOn);
  }

  if(digit=='E'){
    digitalWrite(_A, _segOn);
    digitalWrite(_D, _segOn);
    digitalWrite(_E, _segOn);
    digitalWrite(_F, _segOn);
    digitalWrite(_G, _segOn);
  }

  if(digit=='F'){
    digitalWrite(_A, _segOn);
    digitalWrite(_E, _segOn);
    digitalWrite(_F, _segOn);
    digitalWrite(_G, _segOn);
  }

  if(digit=='G'){
/*
    digitalWrite(_A, _segOn);
    digitalWrite(_B, _segOn);
    digitalWrite(_C, _segOn);
    digitalWrite(_D, _segOn);
    digitalWrite(_E, _segOn);
    digitalWrite(_G, _segOn);
```

```
    // TBD: Really write G like a 9, when it can be written as almost G?
*/
    digitalWrite(_A, _segOn);
    digitalWrite(_C, _segOn);
    digitalWrite(_D, _segOn);
    digitalWrite(_E, _segOn);
    digitalWrite(_F, _segOn);
  }

  if(digit=='H'){
    digitalWrite(_B, _segOn);
    digitalWrite(_C, _segOn);
    digitalWrite(_E, _segOn);
    digitalWrite(_F, _segOn);
    digitalWrite(_G, _segOn);
  }

  if(digit=='I'){
    digitalWrite(_E, _segOn);
    digitalWrite(_F, _segOn);
  }

  if(digit=='J'){
    digitalWrite(_B, _segOn);
    digitalWrite(_C, _segOn);
    digitalWrite(_D, _segOn);
    digitalWrite(_E, _segOn);
  }

  if(digit=='K'){
    digitalWrite(_B, _segOn);
    digitalWrite(_C, _segOn);
    digitalWrite(_E, _segOn);
    digitalWrite(_F, _segOn);
    digitalWrite(_G, _segOn);
  }

  if(digit=='L'){
    digitalWrite(_D, _segOn);
    digitalWrite(_E, _segOn);
    digitalWrite(_F, _segOn);
  }

  if(digit=='M'){
    digitalWrite(_A, _segOn);
    digitalWrite(_C, _segOn);
    digitalWrite(_E, _segOn);
  }

  if(digit=='N'){
    digitalWrite(_C, _segOn);
    digitalWrite(_E, _segOn);
    digitalWrite(_G, _segOn);
  }
```

```
  if(digit=='O'){
    digitalWrite(_A, _segOn);
    digitalWrite(_B, _segOn);
    digitalWrite(_C, _segOn);
    digitalWrite(_D, _segOn);
    digitalWrite(_E, _segOn);
    digitalWrite(_F, _segOn);
  }

  if(digit=='P'){
    digitalWrite(_A, _segOn);
    digitalWrite(_B, _segOn);
    digitalWrite(_E, _segOn);
    digitalWrite(_F, _segOn);
    digitalWrite(_G, _segOn);
  }

  if(digit=='Q'){
    digitalWrite(_A, _segOn);
    digitalWrite(_B, _segOn);
    digitalWrite(_C, _segOn);
    digitalWrite(_F, _segOn);
    digitalWrite(_G, _segOn);
  }

  if(digit=='R'){
    digitalWrite(_E, _segOn);
    digitalWrite(_G, _segOn);
  }

  if(digit=='S'){
    digitalWrite(_A, _segOn);
    digitalWrite(_C, _segOn);
    digitalWrite(_D, _segOn);
    digitalWrite(_F, _segOn);
    digitalWrite(_G, _segOn);
  }

  if(digit=='T'){
    digitalWrite(_D, _segOn);
    digitalWrite(_E, _segOn);
    digitalWrite(_F, _segOn);
    digitalWrite(_G, _segOn);
  }

 if(digit=='U'){
    digitalWrite(_B, _segOn);
    digitalWrite(_C, _segOn);
    digitalWrite(_D, _segOn);
    digitalWrite(_E, _segOn);
    digitalWrite(_F, _segOn);
  }
```

```
  if(digit=='V'){
    digitalWrite(_C, _segOn);
    digitalWrite(_D, _segOn);
    digitalWrite(_E, _segOn);
  }

  if(digit=='W'){
    digitalWrite(_B, _segOn);
    digitalWrite(_D, _segOn);
    digitalWrite(_F, _segOn);
  }

  if(digit=='X'){
    digitalWrite(_B, _segOn);
    digitalWrite(_C, _segOn);
    digitalWrite(_E, _segOn);
    digitalWrite(_F, _segOn);
    digitalWrite(_G, _segOn);
  }

  if(digit=='Y'){
    digitalWrite(_B, _segOn);
    digitalWrite(_C, _segOn);
    digitalWrite(_D, _segOn);
    digitalWrite(_F, _segOn);
    digitalWrite(_G, _segOn);
  }

  if(digit=='Z'){
    digitalWrite(_A, _segOn);
    digitalWrite(_B, _segOn);
    digitalWrite(_D, _segOn);
    digitalWrite(_E, _segOn);
    digitalWrite(_G, _segOn);
  }
}

void SevenSeg::execDelay(int usec){

  if(usec!=0){    // delay() and delayMicroseconds() don't handle 0 delay

    if(usec<=16383)     delayMicroseconds(usec);      // maximum value for
delayMicroseconds();
    else          delay(usec/1000);

  }

}
```

keywords.txt

```
#######################################
# SevenSeg v1.1
# keywords.txt - Syntax Coloring Map
# Sascha Bruechert, 05.02.2015
#######################################

#######################################
# Datatypes (KEYWORD1)
#######################################
SevenSeg    KEYWORD1


#######################################
# Methods and Functions (KEYWORD2)
#######################################
digitPins   KEYWORD2
numOfDigits KEYWORD2

## Low level functions for initializing hardware
setCommonAnode    KEYWORD2
setCommonCathode  KEYWORD2
setDigitPins      KEYWORD2
setActivePinState KEYWORD2
setDPPin    KEYWORD2
setColonPin KEYWORD2
setSymbPins KEYWORD2

## Low level functions for printing to display
clearDisp   KEYWORD2
changeDigit KEYWORD2
writeDigit  KEYWORD2
setDP KEYWORD2
clearDP     KEYWORD2
setColon    KEYWORD2
clearColon  KEYWORD2
setApos     KEYWORD2
clearApos   KEYWORD2

## Low level functions for controlling multiplexing
setDigitDelay     KEYWORD2
setRefreshRate    KEYWORD2
setDutyCycle      KEYWORD2

## High level functions for printing to display
write KEYWORD2
writeClock  KEYWORD2

## Timer control functions
setTimer    KEYWORD2
clearTimer  KEYWORD2
startTimer  KEYWORD2
stopTimer   KEYWORD2
```

```
interruptAction	KEYWORD2

#######################################
# Instances (KEYWORD2)
#######################################

#######################################
# Constants (LITERAL1)
#######################################
```