```
/**
 Configure Interval setting (main screen)
*/
void printIntervalMenu() {

 lcd.setCursor(0, 0);
 lcd.print("Interval      ");
 lcd.setCursor(0, 1);
 if( interval < 20 ){
  lcd.print( printFloat( interval, 5, 1 ) );
 } else {
  lcd.print( printFloat( interval, 3, 0 ) );
 }
 lcd.print( "        " );
}
```

```
/**
 Configure Mode setting
*/
void printModeMenu() {

 lcd.setCursor(0, 0);
 lcd.print("Mode");
 lcd.setCursor(0, 1);
 if ( mode == MODE_M ) {
  lcd.print( "M (Default)    " );
 } else {
  lcd.print( "Bulb (Astro)    " );
 }
}
```

```
/**
 Configure no of shots - 0 means infinity
*/
void printNoOfShotsMenu() {

 lcd.clear();
 lcd.setCursor(0, 0);
 lcd.print("No of shots");
 lcd.setCursor(0, 1);
 if ( maxNoOfShots > 0 ) {
  lcd.print( printInt( maxNoOfShots, 4 ) );
 } else {
  lcd.print( "unlimited" );
 }
 lcd.print( "      "); // clear rest of display
}
```

```
/**
 Configure Exposure setting in Bulb mode
*/
void printExposureMenu() {

 lcd.setCursor(0, 0);
 lcd.print("Exposure       ");
 lcd.setCursor(0, 1);

 int hours = (int)releaseTime / 60 / 60;
 int minutes = ( (int)releaseTime / 60 ) % 60;
 int secs = ( (int)releaseTime ) % 60;
 String sHours = fillZero( hours );
 String sMinutes = fillZero( minutes );
 String sSecs = fillZero( secs );

 lcd.print(releaseTime);
 lcd.print( "          " );
}
```

```
/**
 Print running screen
*/
void printRunningScreen() {

 lcd.setCursor(0, 0);
 lcd.print( printInt( imageCount, 4 ) );

 if ( maxNoOfShots > 0 ) {
  lcd.print( " R:" );
  lcd.print( printInt( maxNoOfShots - imageCount, 4 ) );
  lcd.print( " " );

  lcd.setCursor(0, 1);
  // print remaining time
  unsigned long remainingSecs = (maxNoOfShots - imageCount) * interval;
  lcd.print( "T-");
  lcd.print( fillZero( remainingSecs / 60 / 60 ) );
  lcd.print( ":" );
  lcd.print( fillZero( ( remainingSecs / 60 ) % 60 ) );
 }

 updateTime();

 lcd.setCursor(11, 0);
 if ( millis() < rampingEndTime ) {
  lcd.print( "*" );
 } else {
  lcd.print( " " );
 }
 if( interval < 100 ){ // prevent the interval display from being cut
  lcd.print( printFloat( interval, 4, 1 ) );
 } else {
  lcd.print( printFloat( interval, 4, 0 ) );
 }
}
```

```
void printConfirmEndScreen() {
 lcd.setCursor(0, 0);
 lcd.print( "Stop shooting?");
 lcd.setCursor(0, 1);
 lcd.print( "< Stop    Cont.>");
}
```

```
void printConfirmEndScreenBulb() {
 lcd.setCursor(0, 0);
 lcd.print( "Stop exposure?");
 lcd.setCursor(0, 1);
 lcd.print( "< Cont.   Stop >");
}
```

```
/**
 Print Settings Menu
*/
void printSettingsMenu() {

 lcd.setCursor(1, 0);
 lcd.print("Pause         ");
 lcd.setCursor(1, 1);
 lcd.print("Ramp Interval  ");
 lcd.setCursor(0, settingsSel - 1);
 lcd.print(">");
 lcd.setCursor(0, 1 - (settingsSel - 1));
 lcd.print(" ");
}
```

```
/**
 Pause Mode
*/
void printPauseMenu() {

 lcd.setCursor(0, 0);
 lcd.print("PAUSE...       ");
 lcd.setCursor(0, 1);
 lcd.print("< Continue");
}
```

```
/**
 Print Ramping Duration Menu
*/
void printRampDurationMenu() {

 lcd.setCursor(0, 0);
 lcd.print("Ramp Time (min) ");

 lcd.setCursor(0, 1);
 lcd.print( rampDuration );
 lcd.print( "    " );
}
```

```
/**
 Print Ramping To Menu
*/
void printRampToMenu() {

 lcd.setCursor(0, 0);
 lcd.print("Ramp to (Intvl.)");

 lcd.setCursor(0, 1);
 if( rampTo < 20 ){
  lcd.print( printFloat( rampTo, 5, 1 ) );
 } else {
  lcd.print( printFloat( rampTo, 3, 0 ) );
 }
 lcd.print( "          " );
}
```

```
void printDoneScreen() {

 // print elapsed image count))
 lcd.setCursor(0, 0);
 lcd.print("Done ");
 lcd.print( imageCount );
 lcd.print( " shots.");

 // print elapsed time when done
 lcd.setCursor(0, 1);
 lcd.print( "t=");
 lcd.print( fillZero( runningTime / 1000 / 60 / 60 ) );
 lcd.print( ":" );
 lcd.print( fillZero( ( runningTime / 1000 / 60 ) % 60 ) );
 lcd.print("    ;-)");
}
```

```
/**
 Update the time display in the main screen
*/
void updateTime() {

 unsigned long finerRunningTime = runningTime + (millis() - previousMillis);

 if ( isRunning ) {

  int hours = finerRunningTime / 1000 / 60 / 60;
  int minutes = (finerRunningTime / 1000 / 60) % 60;
  int secs = (finerRunningTime / 1000 ) % 60;

  String sHours = fillZero( hours );
  String sMinutes = fillZero( minutes );
  String sSecs = fillZero( secs );

  lcd.setCursor(8, 1);
  lcd.print( sHours );
  lcd.setCursor(10, 1);
  lcd.print(":");
  lcd.setCursor(11, 1);
  lcd.print( sMinutes );
  lcd.setCursor(13, 1);
  lcd.print(":");
  lcd.setCursor(14, 1);
  lcd.print( sSecs );
 } else {
  lcd.setCursor(8, 1);
  lcd.print("   Done!");
 }
}
```

```
void printSingleScreen() {
 lcd.setCursor(0, 0);

 if ( releaseTime < 1 ) {
  lcd.print( "Single Exposure");
  lcd.setCursor(0, 1);
  lcd.print(releaseTime); // under one second
  lcd.print("      ");
 } else {
  lcd.print( "Bulb Exposure  ");
  lcd.setCursor(0, 1);

  if ( bulbReleasedAt == 0 ) { // if not shooting
   // display exposure time setting
   int hours = (int)releaseTime / 60 / 60;
   int minutes = ( (int)releaseTime / 60 ) % 60;
   int secs = ( (int)releaseTime ) % 60;
   String sHours = fillZero( hours );
   String sMinutes = fillZero( minutes );
   String sSecs = fillZero( secs );

   lcd.print( sHours );
   lcd.print(":");
   lcd.print( sMinutes );
   lcd.print( "\'");
   lcd.print( sSecs );
   lcd.print( "\"");
   lcd.print( " " );
  }
 }

 if ( bulbReleasedAt == 0 ) { // currently not running

  lcd.setCursor(10, 1);
  lcd.print( "< FIRE" );

 } else { // running
  unsigned long runningTime = ( bulbReleasedAt + releaseTime * 1000 ) - millis();

  int hours = runningTime / 1000 / 60 / 60;
  int minutes = ( runningTime / 1000 / 60 ) % 60;
  int secs = ( runningTime / 1000 ) % 60;

  String sHours = fillZero( hours );
  String sMinutes = fillZero( minutes );
  String sSecs = fillZero( secs );

  lcd.setCursor(0, 1);
  lcd.print("      "); // clear time setting display
  lcd.setCursor(8, 1);
  lcd.print( sHours );
  lcd.setCursor(10, 1);
  lcd.print(":");
  lcd.setCursor(11, 1);
  lcd.print( sMinutes );
  lcd.setCursor(13, 1);
  lcd.print(":");
  lcd.setCursor(14, 1);
  lcd.print( sSecs );
 }
}
```

```
void printScreen() {

 switch ( currentMenu ) {

  case SCR_INTERVAL:
   printIntervalMenu();
   break;

  case SCR_MODE:
   printModeMenu();
   break;

  case SCR_SHOTS:
   printNoOfShotsMenu();
   break;

  case SCR_EXPOSURE:
   printExposureMenu();
   break;

  case SCR_RUNNING:
   printRunningScreen();
   break;

  case SCR_CONFIRM_END:
   printConfirmEndScreen();
   break;

  case SCR_CONFIRM_END_BULB:
   printConfirmEndScreenBulb();
   break;

  case SCR_SETTINGS:
   printSettingsMenu();
   break;

  case SCR_PAUSE:
   printPauseMenu();
   break;

  case SCR_RAMP_TIME:
   printRampDurationMenu();
   break;

  case SCR_RAMP_TO:
   printRampToMenu();
   break;

  case SCR_DONE:
   printDoneScreen();
   break;

  case SCR_SINGLE:
   printSingleScreen();
   break;

 }
}
```

LCD values

99

639 408 0 1023

255

```
// ---------- HELPER METHODS ----------------------------------

/**
 Fill in leading zero to numbers in order to always have 2 digits
*/
String fillZero( int input ) {

 String sInput = String( input );
 if ( sInput.length() < 2 ) {
  sInput = "0";
  sInput.concat( String( input ));
 }
 return sInput;
}

String printFloat(float f, int total, int dec) {

 static char dtostrfbuffer[8];
 String s = dtostrf(f, total, dec, dtostrfbuffer);
 return s;
}

String printInt( int i, int total) {
 float f = i;
 static char dtostrfbuffer[8];
 String s = dtostrf(f, total, 0, dtostrfbuffer);
 return s;
}
```

```
/**
 Slider initialization to home position = 0
*/
void Rail_ini()
{

 if (digitalRead(TOGG)== LOW){
   iniOver = - iniOver; //-200
   iniHome = 0;
   iniHunt = - iniHunt;
   }
  if ( digitalRead(TOGG)== HIGH){
   iniOver = iniOver + endPos ; //21200
   iniHome = endPos; //21000
   iniHunt = iniHunt;
   }

 stepper.setMaxSpeed(800);
 stepper.setAcceleration(200);

 while (digitalRead(LIMIT) == HIGH){
  stepper.moveTo(iniHunt);  //move slowly in the direction switch till it hits the limit switch
  stepper.run();
 }
 stepper.setCurrentPosition(iniOver); // Put current position to overshoot end position
 while (stepper.currentPosition() != iniHome)
 {
  stepper.moveTo(iniHome);  //Move slowy on the other direction to clear the limit switch and put to 0 or 126000
  stepper.run();
 }
 }
```

```
/**  Slider Manual move
     Short push = inch
     Long push = middle, Second long = end of rail
*/
void Inch_Man()
{
 inchval = digitalRead(INCH);
 toggval = digitalRead(TOGG);
 hBval = digitalRead(LIMIT);

 stepper.setMaxSpeed(800);
 stepper.setAcceleration(200);

while (digitalRead(INCH)== HIGH && SteppInch == false && hBval == HIGH){
  delay(100);  //if you want more resolution, lower this number
  pressLength_milliSeconds = pressLength_milliSeconds + 100;
    Serial.println(pressLength_milliSeconds);
 }

if (pressLength_milliSeconds >= optionTwo_milliSeconds){
 SteppInch = true;
 TargetCal = true;
  if (TargetCal == true && toggval == HIGH){
  if (stepper.currentPosition() >= homePos && stepper.currentPosition() < midPos)
  {
   target = midPos;
  }
  else if (stepper.currentPosition() >= midPos && stepper.currentPosition() < endPos)
  {
   target = endPos;
   }
  }
  else if (TargetCal == true && toggval == LOW){
   if (stepper.currentPosition() <= endPos && stepper.currentPosition() > midPos)
   {
   target = midPos;
   }
   else if (stepper.currentPosition() <= midPos && stepper.currentPosition() > homePos)
   {
   target = homePos;
   }
  }
 TargetCal = false;
 if (SteppInch = true)
 {
 stepper.moveTo(target);
 while (stepper.currentPosition() != target)
 {
   if (digitalRead(LIMIT)== HIGH){
   stepper.run();
   } else {
    break;
  }
 }
 SteppInch = false;
 }
}
 else if(pressLength_milliSeconds >= optionOne_milliSeconds){

if (digitalRead(TOGG)== HIGH){

 curPos = stepper.currentPosition ();
 targetInch = curPos + inchMoveval;

stepper.moveTo (target);

 while (stepper.currentPosition () != targetInch){
  if (digitalRead(LIMIT)== HIGH){
  stepper.setSpeed(400);
  stepper.runSpeed();
   }
  }
 }

if ( digitalRead(TOGG)== LOW){

 curPos = stepper.currentPosition ();
 targetInch = curPos - inchMoveval;

 stepper.moveTo (target);

 while (stepper.currentPosition () != targetInch){
  if (digitalRead(LIMIT)== HIGH){
  stepper.setSpeed(-400);
  stepper.runSpeed();
   }
  }
 }
}
pressLength_milliSeconds = 0;
}
```

```
/**
  Quad Ramp calculation

IF(t/(d/2)<=1 { c/2*(t/(d/2))*(t/(d/2))+b }
else{ -c/2*(((2-((t/(d/2))-1))/2)*((2-((t/(d/2))-1))/2)*((2-((t/(d/2))-1))/2)*((2-((t/(d/2))-1))/2)*((2-((t/(d/2))-1))/2)-2)+b  }

*/

void Ramp_Calc(){

Serial.println("Calculation");

Serial.println(imageCount);

duration = maxNoOfShots * ratio;
int change = endPos/maxNoOfShots ; // Scope of the change = c

float t = imageCount;
float b = begining;
float c = change;
float d = duration;

if (imageCount <= duration){

 float ratio_td = t/(d/2);

 //Ramp at Start
 if (ratio_td <= 1){
  oneMove = c/2*ratio_td*ratio_td+b;
 }
 else
 {
 oneMove = -c/2*(((2-((ratio_td)-1))/2)*((2-((ratio_td)-1))/2)*((2-((ratio_td)-1))/2)*((2-((ratio_td)-1))/2)*((2-((ratio_td)-1))/2)-2)+b;
 }

 oneMove = min(oneMove, change);
}
else
{
 // Ramp at end

 if (imageCount >= maxNoOfShots - duration){

  int countDown = imageCount-(maxNoOfShots-duration);
  float tt = countDown;
  b = change;

  float ratio_td = tt/(d/2);

   if (ratio_td <= 1){
   oneMove = c/2*ratio_td*ratio_td-b;
   oneMove = - oneMove;
   }
   else
   {
   oneMove = -c/2*(((2-((ratio_td)-1))/2)*((2-((ratio_td)-1))/2)*((2-((ratio_td)-1))/2)*((2-((ratio_td)-1))/2)*((2-((ratio_td)-1))/2)-2)-b;
   oneMove = - oneMove;
   }
   oneMove = max(oneMove,0);
 }
 else
 {
  //Ramp move for the rest of the time
  oneMove = change;
 }
}
 oneMove = min(oneMove,maxLength); //move no more then maxLength

 Serial.println(oneMove);
 Serial.println("End Cal");
}
```

```
/**
  Slave mode: When a pulse from Camera move a certain distance
*/

void Rail_Slave(){

 Ramp_Calc();

  Serial.println(oneMove);

if (digitalRead(TOGG)== HIGH){

 stepper.move(oneMove);  // Distance of one slider's move

 while (stepper.distanceToGo() != 0){
   if (digitalRead(LIMIT)== HIGH){
  stepper.setSpeed(500);
  stepper.runSpeed();
  }
 }
}

if ( digitalRead(TOGG)== LOW){

curPos = stepper.currentPosition ();
target = curPos - oneMove;

stepper.moveTo(target);  // Distance of one slider's move

 while (stepper.currentPosition () != target){
   if (digitalRead(LIMIT)== HIGH){
  stepper.setSpeed(-500);
  stepper.runSpeed();
  }
 }
}
}
```

```
/*
 K.H: dimming LCD BAckground light
*/
void DimLCD( byte startval, byte endval, byte stepdelay) {
 if (endval < startval) {
  for ( int bl = startval; bl >= endval; bl--) {
   analogWrite(BACK_LIGHT, bl);   // dimming backlight off.
   delay(stepdelay);
  }
 }
 else {
  for ( int bl = startval; bl <= endval; bl++) {
   analogWrite(BACK_LIGHT, bl);   // dimming backlight off.
   delay(stepdelay);
  }
 }
}
```

```
/**
 Will be called by the loop and check if a bulb exposure has to end. If so, it will stop the exposure.
*/
void possiblyEndLongExposure() {
 if ( ( bulbReleasedAt != 0 ) && ( millis() >= ( bulbReleasedAt + releaseTime * 1000 ) ) ) {
  bulbReleasedAt = 0;
  digitalWrite(12, LOW);
 }

 if ( currentMenu == SCR_RUNNING ) { // display exposure indicator on running screen only
  if ( bulbReleasedAt == 0 ) {
   lcd.setCursor(7, 1);
   lcd.print( " ");
  } else {
   lcd.setCursor(7, 1);
   lcd.print((char)255);
  }
 }

 if ( currentMenu == SCR_SINGLE ) {
  printSingleScreen();
 }
}
```

```
/*Berdine Time Lapse Rail
 1) Void Rail_ini to put the slidder at home position depending on toggle switch direction
 2) Void Inch_Man to inch OR send the slider to middle - end or home position:
   The push button has two functions:
 a) Short pusch (less then one secound)= smal move - inching
 b) By pushing more then one second the slider goes to the middle,
      pushing a secound time send the slider to the end.
 3) Void Ramp_Calc is a Quad easyramping IN and OUT at begening and end of rail
 4) If the slider touch the limitswitch it stop.
*/

#include <AccelStepper.h>

AccelStepper stepper(1, A5, A4); // 1 = Easy Driver, A5 STEP pin, A4 DIR pin

const int LIMIT = A1; // Limit switch connected to pin A1
const int SHUT_PIN = 3;  //Shutter cable
const int SLAVE_PIN = 11; //Input signal from the camera hot shoe
const int TOGG = 12;  // Toggle switch connected to pin 12
const int INCH = 13;  // Inch Push buton connected to pin 13

//*** Ramp variables ***
int begining = 0; //value of oneMove at the begining of the ramp = b
int oneMove; // Distance of one slider's move
int duration; //How many frames the ramping is lasting for = d
int y = 1; // Variable to store last 20% of ramping
float ratio = 0.2;  //How long last the ramping in % (0.2 = 20%)
const int maxLength = 100; //Set the maximum lenght for one move of the slider

// *** Inch_Man variables ***
int SteppInch = false;  // logic condition to run manual
int TargetCal = false;  // logic condition to calculate target
long curPos;    //current position
long target;    //move to position variable
long targetInch; //set new target position for inch

// *** Enter here variable depending on rail length ***
long homePos = 0;  //home position
long midPos = 10500;   //half way: depends on rail's length
long endPos = 21000;  //End of rail: depends on rail's length
int inchMoveval = 200; //When pushbutton short value of one move

//Variables to set the ini direction
long iniOver = inchMoveval + 200;
long iniHunt = endPos + 1000;
long iniHome;

//Variable for Dual push button function
float pressLength_milliSeconds = 0;
int optionOne_milliSeconds = 100;   //Short push
int optionTwo_milliSeconds = 1000;  //Long push
```

```
/*
 Pro-Timer Free
 Gunther Wegner
 http://gwegner.de
 http://lrtimelapse.com
 https://github.com/gwegner/LRTimelapse-Pro-Timer-Free

 Version 0.89: Klaus Heiss: Lcd backlight dimming implemented, Save Params in EEPROM
implemented (lib EEPROMConfig) 28.08.17
 Version 0.88: Thanks for Klaus Heiss (KH) for implementing the dynamic key rate
*/


#include <LiquidCrystal.h>
#include "LCD_Keypad_Reader.h"          // credits to: http://www.hellonull.com/?p=282
#include "EEPROMConfig.h"  // Klaus Heiss, www.elite.at

const String CAPTION = "Pro-Timer 0.91";

LCD_Keypad_Reader keypad;
LiquidCrystal lcd(8, 9, 4, 5, 6, 7); //Pin assignments for SainSmart LCD Keypad Shield

const int NONE = 0;              // Key constants
const int SELECT = 1;
const int LEFT = 2;
const int UP = 3;
const int DOWN = 4;
const int RIGHT = 5;

const int Onboard_LED = 13;
const int BACK_LIGHT  = 10;

const float RELEASE_TIME_DEFAULT = 0.02;    // default shutter release time for camera
const float MIN_DARK_TIME = 0.5;

const int keyRepeatRate = 100; // when held, key repeats 1000 / keyRepeatRate times per second

const int decoupleTime = 1000; // time in milliseconds to wait before doing a single bulb exposure

int localKey = 0; // The current pressed key
int lastKeyPressed = -1; // The last pressed key

unsigned long lastKeyCheckTime = 0;
unsigned long lastKeyPressTime = 0;
unsigned long lastDispTurnedOffTime = 0;

int sameKeyCount = 0;
float releaseTime = RELEASE_TIME_DEFAULT; // Shutter release time for camera
unsigned long previousMillis = 0; // Timestamp of last shutter release
unsigned long runningTime = 0;

float interval = 4.0;              // the current interval
long maxNoOfShots = 0;
int isRunning = 0; // flag indicates intervalometer is running
unsigned long bulbReleasedAt = 0;

int imageCount = 0;  // Image count since start of intervalometer

unsigned long rampDuration = 10; // ramping duration
float rampTo = 0.0; // ramping interval
unsigned long rampingStartTime = 0; // ramping start time
unsigned long rampingEndTime = 0; // ramping end time
float intervalBeforeRamping = 0; // interval before ramping

boolean backLight = HIGH; // The current settings for the backlight

const int SCR_INTERVAL = 0; // menu workflow constants
const int SCR_SHOTS = 1;
const int SCR_MODE = 9;
const int SCR_EXPOSURE = 10;
const int SCR_RUNNING = 2;
const int SCR_CONFIRM_END = 3;
const int SCR_CONFIRM_END_BULB = 12;
const int SCR_SETTINGS = 4;
const int SCR_PAUSE = 5;
const int SCR_RAMP_TIME = 6;
const int SCR_RAMP_TO = 7;
const int SCR_DONE = 8;

// left side MENUS
const int SCR_SINGLE = 11;

const int MODE_M = 0;
const int MODE_BULB = 1;


int currentMenu = 0; // the currently selected menu
int settingsSel = 1; // the currently selected settings option
int mode = MODE_M;  // mode: M or Bulb

const float cMinInterval = 0.2;
const float cMaxInterval = 999;  // no intervals longer as 999secs - those would scramble the display

// K.H. LCD dimming
const int cMinLevel    = 0;  // Min. Background Brightness Levels
const int cMaxLevel    = 5;  // Max. Background Brightness Levels
int  act_BackLightLevel = 4;
char act_BackLightDir   = 'D';

// K.H: EPROM Params
EEPParams EEProm;
```

```
/**
 Initialize everything
*/
void setup() {
 pinMode(Onboard_LED, OUTPUT);
 digitalWrite(Onboard_LED, LOW);   // Turn Onboard LED OFF. it only consumes battery power ;-)

 pinMode(BACK_LIGHT, OUTPUT);
 digitalWrite(BACK_LIGHT, LOW);    // First turn backlight off.

 Serial.begin(9600);
 Serial.println( CAPTION );

 // init LCD display
 lcd.begin(16, 2);
 lcd.clear();
 lcd.setCursor(0, 0);
 // print welcome screen))
 lcd.print("LRTimelapse.com");
 lcd.setCursor(0, 1);
 lcd.print( CAPTION );

 // Load EEPROM params
 EEProm.ParamsRead();
 // check ranges
 EEProm.Params.BackgroundBrightnessLevel =
constrain(EEProm.Params.BackgroundBrightnessLevel, cMinLevel,    cMaxLevel);
 EEProm.Params.Interval           = constrain(EEProm.Params.Interval,            cMinInterval,
cMaxInterval);

 act_BackLightLevel = EEProm.Params.BackgroundBrightnessLevel;
 interval         = EEProm.Params.Interval;

 // wait a moment...  show CAPTION dimming
 /* H.K.: implemented dimming */
 digitalWrite(BACK_LIGHT, HIGH);    // Turn backlight on.
 delay(2000);
 DimLCD(255,act_BackLightBrightness(),2);

  //Rail_ini mode
  pinMode(LIMIT, INPUT_PULLUP); //Pin mode for the Limit switch
  pinMode(TOGG,INPUT);   //Pin mode for the direction toggle switch

  Rail_ini();

  //Manual Mode
  pinMode(INCH,INPUT);  //Pin mode for the inch push buttom

  //Slave mode
  pinMode(SHUT_PIN, OUTPUT);  // initialize output pin for camera release
  pinMode(SLAVE_PIN, INPUT_PULLUP); // sets the digital pin as input for the camera signal

 printIntervalMenu();

 lcd.clear();
}
```

```
/**
 The main loop
*/
void loop() {

 Inch_Man();

  if (isRunning == 1){

   if (digitalRead(SLAVE_PIN)== LOW){

   Rail_Slave();
   }
  }
 if (millis() > lastKeyCheckTime + keySampleRate) {
  lastKeyCheckTime = millis();
  localKey = keypad.getKey();
  //Serial.println( localKey );
  //Serial.println( lastKeyPressed );

  if (localKey != lastKeyPressed) {
   processKey();
   keypad.RepeatRate = keyRepeatRateSlow;
  } else {
   // key value has not changed, key is being held down, has it been long enough?
   // (but don't process localKey = 0 = no key pressed)

   /* H.K.: implemented function ActRepeateRate instead of constant */
   if (localKey != 0 && millis() > lastKeyPressTime + keypad.ActRepeatRate()) {
    // yes, repeat this key
    if ( (localKey == UP ) || ( localKey == DOWN ) || ( localKey == SELECT ) ) {
     processKey();
    }
   }
  }

  if ( currentMenu == SCR_RUNNING ) {
   printScreen();               // update running screen in any case

   if ( mode == MODE_BULB ) {
    possiblyEndLongExposure(); // checks regularly if a long exposure has to be cancelled
   }
  }

  if ( currentMenu == SCR_SINGLE ) {
   possiblyEndLongExposure();
  }
 }
 if ( isRunning ) {               // release camera, do ramping if running
  running();
 }
}
```

```
void firstShutter() {

 lcd.clear();
 lcd.setCursor(0,0);
 lcd.print("Decoupling...");
 delay( decoupleTime );

 previousMillis = millis();
 runningTime = 0;
 isRunning = 1;

 lcd.clear();
 printRunningScreen();

 // do the first release instantly, the subsequent ones will happen in the loop
 releaseCamera();
 imageCount++;
}
```

```
void stopShooting() {
 isRunning = 0;
 imageCount = 0;
 runningTime = 0;
 bulbReleasedAt = 0;
}
```

```
/**
 Actually release the camera
*/
void releaseCamera() {

 // short trigger in M-Mode
 if ( releaseTime < 1 ) {
  if ( currentMenu == SCR_RUNNING ) { // display exposure indicator on running screen only
   lcd.setCursor(7, 1);
   lcd.print((char)255);
  }

  digitalWrite(SHUT_PIN, HIGH); ; // ****replace digitalWrite(12, HIGH)****
  delay( releaseTime * 1000 );
  digitalWrite(SHUT_PIN, LOW); // ****replace digitalWrite(12, LOW)****

  if ( currentMenu == SCR_RUNNING ) { // clear exposure indicator
   lcd.setCursor(7, 1);
   lcd.print(" ");
  }

 } else { // releaseTime > 1 sec

  // long trigger in Bulb-Mode for longer exposures
  if ( bulbReleasedAt == 0 ) {
   bulbReleasedAt = millis();
   digitalWrite(SHUT_PIN, HIGH); // ****replace digitalWrite(12, HIGH)****
  }
 }
}
```

```
/**
 Running, releasing Camera
*/
void running() {

 // do this every interval only
 if ( ( millis() - previousMillis ) >=  ( ( interval * 1000 )) ) {

  if ( ( maxNoOfShots != 0 ) && ( imageCount >= maxNoOfShots ) ) { // sequence is finished
   // stop shooting
   isRunning = 0;
   currentMenu = SCR_DONE;
   lcd.clear();
   printDoneScreen(); // invoke manually
   stopShooting();
  } else { // is running
   runningTime += (millis() - previousMillis );
   previousMillis = millis();
   releaseCamera();
   imageCount++;
  }
 }

 // do this always (multiple times per interval)
 possiblyRampInterval();
}
```

```
/**
 If ramping was enabled do the ramping
*/
void possiblyRampInterval() {

 if ( ( millis() < rampingEndTime ) && ( millis() >= rampingStartTime ) ) {
  interval = intervalBeforeRamping + ( (float)( millis() - rampingStartTime ) / (float)( rampingEndTime - rampingStartTime ) * ( rampTo - intervalBeforeRamping ) );

  if ( releaseTime > interval - MIN_DARK_TIME ) { // if ramping makes the interval too short for the exposure time in bulb mode, adjust the exposure time
   releaseTime =  interval - MIN_DARK_TIME;
  }

 } else {
  rampingStartTime = 0;
  rampingEndTime = 0;
 }
}
```

```
/**
 Process the key presses - do the Menu Navigation
*/
void processKey() {

 // select key will switch backlight at any time
 if ( localKey == SELECT ) {
  if (lastKeyPressed == SELECT) {
   if (millis() > lastKeyPressTime + 500) {
    changeBackLightBrightness('R');
    lastKeyPressTime = millis();
    backLight = true;
   }
  }
  else {
   backLight = !backLight;
   if (backLight) {
    analogWrite(BACK_LIGHT, act_BackLightBrightness()); // Turn PWM backlight on.
    if (millis() < lastDispTurnedOffTime + 450) {
     save_Params();
     lastDispTurnedOffTime = 0;
    }
   }
   else {
    digitalWrite(BACK_LIGHT, LOW); // Turn backlight off.
    lastDispTurnedOffTime = millis();
   }
   lastKeyPressed = localKey;
   lastKeyPressTime = millis();
  }
 }
 else {
  lastKeyPressed = localKey;
  lastKeyPressTime = millis();
 }

 // do the menu navigation
 switch ( currentMenu ) {

  case SCR_INTERVAL:

   if ( localKey == UP ) {
    if( interval < 20 ){
     interval = (float)((int)(interval * 10) + 1) / 10; // round to 1 decimal place
    } else {
     interval = (float)((int)interval + 1); // round to 1 decimal place
    }
    if ( interval > cMaxInterval ) {
     interval = cMaxInterval;
    }
   }

   if ( localKey == DOWN ) {
    if ( interval > cMinInterval) {
     if( interval < 20 ){
      interval = (float)((int)(interval * 10) - 1) / 10; // round to 1 decimal place
     } else {
      interval = (float)((int)interval - 1);
     }
    }
   }

   if ( localKey == RIGHT ) {
    lcd.clear();
    rampTo = interval;          // set rampTo default to the current interval
    currentMenu = SCR_MODE;
   }

   if ( localKey == LEFT ) {
    lcd.clear();
    currentMenu = SCR_SINGLE;
   }

   break;

  case SCR_MODE:
   if ( localKey == RIGHT ) {
    currentMenu = SCR_SHOTS;
   }
   else if ( localKey == LEFT ) {
    currentMenu = SCR_INTERVAL;
   }
   else if ( ( localKey == UP ) || ( localKey == DOWN ) ) {
    if ( mode == MODE_M ) {
     mode = MODE_BULB;
    } else {
     mode = MODE_M;
     releaseTime = RELEASE_TIME_DEFAULT;   // when switching to M-Mode, set the shortest shutter release time.
    }
   }
   break;

  case SCR_SHOTS:

   if ( localKey == UP ) {
    if ( maxNoOfShots >= 2500 ) {
     maxNoOfShots += 100;
    } else if ( maxNoOfShots >= 1000 ) {
     maxNoOfShots += 50;
    } else if ( maxNoOfShots >= 100 ) {
     maxNoOfShots += 25;
    } else if ( maxNoOfShots >= 10 ) {
     maxNoOfShots += 10;
    } else {
     maxNoOfShots ++;
    }
    if ( maxNoOfShots >= 9999 ) { // prevents screwing the ui
     maxNoOfShots = 9999;
    }
   }

   if ( localKey == DOWN ) {
    if ( maxNoOfShots > 2500 ) {
     maxNoOfShots -= 100;
    } else if ( maxNoOfShots > 1000 ) {
     maxNoOfShots -= 50;
    } else if ( maxNoOfShots > 100 ) {
     maxNoOfShots -= 25;
    } else if ( maxNoOfShots > 10 ) {
     maxNoOfShots -= 10;
    } else if ( maxNoOfShots > 0) {
     maxNoOfShots -= 1;
    } else {
     maxNoOfShots = 0;
    }
   }

   if ( localKey == LEFT ) {
    currentMenu = SCR_MODE;
    stopShooting();
    lcd.clear();
   }

   if ( localKey == RIGHT ) { // Start shooting
    if ( mode == MODE_M ) {
     currentMenu = SCR_RUNNING;
     firstShutter();
    } else {
     currentMenu = SCR_EXPOSURE; // in Bulb mode ask for exposure
    }
   }
   break;

  // Exposure Time
  case SCR_EXPOSURE:

   if ( localKey == UP ) {
    releaseTime = (float)((int)(releaseTime * 10) + 1) / 10; // round to 1 decimal place
   }

   if ( releaseTime > interval - MIN_DARK_TIME ) { // no release times longer then (interval-Min_Dark_Time)
    releaseTime = interval - MIN_DARK_TIME;
   }

   if ( localKey == DOWN ) {
    if ( releaseTime > RELEASE_TIME_DEFAULT ) {
     releaseTime = (float)((int)(releaseTime * 10) - 1) / 10; // round to 1 decimal place
    }
   }

   if ( localKey == LEFT ) {
    currentMenu = SCR_SHOTS;
   }

   if ( localKey == RIGHT ) {
    currentMenu = SCR_RUNNING;
    firstShutter();
   }

   break;

  case SCR_RUNNING:

   if ( localKey == LEFT ) { // LEFT from Running Screen aborts

    if ( rampingEndTime == 0 ) {  // if ramping not active, stop the whole shot, other otherwise only the ramping
     if ( isRunning ) {  // if is still runing, show confirm dialog
      currentMenu = SCR_CONFIRM_END;
     } else {  // if finished, go to start screen
      currentMenu = SCR_INTERVAL;
     }
     lcd.clear();
    } else { // stop ramping
     rampingStartTime = 0;
     rampingEndTime = 0;
    }

   }

   if ( localKey == RIGHT ) {
    currentMenu = SCR_SETTINGS;
    lcd.clear();
   }

   if ( localKey == UP ) {
    changeBackLightBrightness('U');
   }
   if ( localKey == DOWN ) {
    changeBackLightBrightness('D');
   }
   break;

  case SCR_CONFIRM_END:
   if ( localKey == LEFT ) { // Really abort
    currentMenu = SCR_INTERVAL;

    if ( bulbReleasedAt > 0 ) { // if we are shooting in bulb mode, instantly stop the exposure
     bulbReleasedAt = 0;
     digitalWrite(12, LOW);
    }
    stopShooting();
    lcd.clear();
   }
   if ( localKey == RIGHT ) { // resume
    currentMenu = SCR_RUNNING;
    lcd.clear();
   }
   break;

  case SCR_SETTINGS:

   if ( localKey == DOWN && settingsSel == 1 ) {
    settingsSel = 2;
   }

   if ( localKey == UP && settingsSel == 2 ) {
    settingsSel = 1;
   }

   if ( localKey == LEFT ) {
    settingsSel = 1;
    currentMenu = SCR_RUNNING;
    lcd.clear();
   }

   if ( localKey == RIGHT && settingsSel == 1 ) {
    isRunning = 0;
    currentMenu = SCR_PAUSE;
    lcd.clear();
   }

   if ( localKey == RIGHT && settingsSel == 2 ) {
    currentMenu = SCR_RAMP_TIME;
    lcd.clear();
   }
   break;

  case SCR_PAUSE:

   if ( localKey == LEFT ) {
    currentMenu = SCR_RUNNING;
    isRunning = 1;
    previousMillis = millis() - (imageCount * 1000); // prevent counting the paused time as running time;
    lcd.clear();
   }
   break;

  case SCR_RAMP_TIME:

   if ( localKey == RIGHT ) {
    currentMenu = SCR_RAMP_TO;
    lcd.clear();
   }

   if ( localKey == LEFT ) {
    currentMenu = SCR_SETTINGS;
    settingsSel = 2;
    lcd.clear();
   }

   if ( localKey == UP ) {
    if ( rampDuration >= 10) {
     rampDuration += 10;
    } else {
     rampDuration += 1;
    }
   }

   if ( localKey == DOWN ) {
    if ( rampDuration > 10 ) {
     rampDuration -= 10;
    } else {
     rampDuration -= 1;
    }
    if ( rampDuration <= 1 ) {
     rampDuration = 1;
    }
   }
   break;

  case SCR_RAMP_TO:

   if ( localKey == LEFT ) {
    currentMenu = SCR_RAMP_TIME;
    lcd.clear();
   }

   if ( localKey == UP ) {
    if( rampTo < 20 ){
     rampTo = (float)((int)(rampTo * 10) + 1) / 10; // round to 1 decimal place
    } else {
     rampTo = (float)((int)rampTo + 1); // round to 1 decimal place
    }
    if ( rampTo > cMaxInterval ) {
     rampTo = cMaxInterval;
    }
   }

   if ( localKey == DOWN ) {
    if ( rampTo > cMinInterval) {
     if( rampTo < 20 ){
      rampTo = (float)((int)(rampTo * 10) - 1) / 10; // round to 1 decimal place
     } else {
      rampTo = (float)((int)rampTo - 1);
     }
    }
   }

   if ( localKey == RIGHT ) { // start Interval ramping
    if ( rampTo != interval ) { // only if a different Ramping To interval has been set!
     intervalBeforeRamping = interval;
     rampingStartTime = millis();
     rampingEndTime = rampingStartTime + rampDuration * 60 * 1000;
    }

    // go back to main screen
    currentMenu = SCR_RUNNING;
    lcd.clear();
   }
   break;

  case SCR_DONE:

   if ( localKey == LEFT || localKey == RIGHT ) {
    currentMenu = SCR_INTERVAL;
    stopShooting();
    lcd.clear();
   }
   break;

  case SCR_SINGLE:
   if ( localKey == UP ) {
    if ( releaseTime < 60 )
     releaseTime = (float)((int)(releaseTime + 1));
    else
     releaseTime = (float)((int)(releaseTime + 10));
   }

   if ( localKey == DOWN ) {
    if ( releaseTime > RELEASE_TIME_DEFAULT ) {
     if ( releaseTime < 60 )
      releaseTime = (float)((int)(releaseTime - 1));
     else
      releaseTime = (float)((int)(releaseTime - 10));
    }
    if ( releaseTime < RELEASE_TIME_DEFAULT ) { // if it's too short after decrementing, set to the default release time.
     releaseTime = RELEASE_TIME_DEFAULT;
    }
   }

   if ( localKey == LEFT ) {
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("Decoupling...");
    delay( decoupleTime );
    lcd.clear();
    releaseCamera();
   }

   if ( localKey == RIGHT ) {
    lcd.clear();
    if ( bulbReleasedAt == 0 ) { // if not running, go to main screen
     currentMenu = SCR_INTERVAL;
    } else {             // if running, go to confirm screen
     currentMenu = SCR_CONFIRM_END_BULB;
    }

   }
   break;

  case SCR_CONFIRM_END_BULB:
   if ( localKey == RIGHT ) { // Really abort

    digitalWrite(12, LOW);
    stopShooting();

    currentMenu = SCR_SINGLE;
    lcd.clear();
   }
   if ( localKey == LEFT ) { // resume
    currentMenu = SCR_SINGLE;
    lcd.clear();
   }
   break;

 }
 printScreen();
}
```

```
/**
 K.H: Change Backlight Brightness in Steps
*/
void changeBackLightBrightness( char AMode) {   //U=up, D=douwn, R=rolling
 if ((AMode == 'U') || (AMode == 'D')) {
  act_BackLightDir = AMode;
 }
 if (act_BackLightDir == 'D') {
  act_BackLightLevel --;
 } else {
  act_BackLightLevel ++;
 }

 if ((act_BackLightLevel == -1) && (AMode == 'R')) {
  act_BackLightDir = 'U';
 }
 if ((act_BackLightLevel == cMaxLevel + 1)  && (AMode == 'R')) {
  act_BackLightDir = 'D';
 }
 if (AMode == 'R') {
  act_BackLightLevel = constrain(act_BackLightLevel, cMinLevel - 1, cMaxLevel + 1);
 } else {
  act_BackLightLevel = constrain(act_BackLightLevel, cMinLevel, cMaxLevel);
 }

 analogWrite(BACK_LIGHT, act_BackLightBrightness()); // Turn PWM backlight on.
}
```

```
/**
 K.H. calc PWM-Value for background display brightness
*/
byte act_BackLightBrightness() {
 return constrain (act_BackLightLevel * (255 / cMaxLevel), 12, 255);
}
```

```
/**
  K.H. save actual level in EPROM
*/
void save_Params() {
 String str;
 if (not isRunning) {
  boolean save = false;
  act_BackLightLevel = constrain(act_BackLightLevel, cMinLevel, cMaxLevel);
  if (act_BackLightLevel != EEProm.Params.BackgroundBrightnessLevel) {
   str = "saved level "; str += (act_BackLightLevel + 1); str += "   ";
   lcd.setCursor(0, 0);
   lcd.print(str);
   delay(2000);
   save = true;
  };
  if (interval != EEProm.Params.Interval) {
   str = "saved int. "; str += String(interval, 2); str += "  ";
   lcd.setCursor(0, 0);
   lcd.print(str);
   delay(2000);
   save = true;
  };
  if (save) {
   EEProm.Params.BackgroundBrightnessLevel = act_BackLightLevel;
   EEProm.Params.Interval              = interval;
   if (not EEProm.ParamsWrite()) {
    lcd.setCursor(0, 0); lcd.print("PROBLEM SAVING  ");
    lcd.setCursor(0, 1); lcd.print("PARAMS TO EEPROM");
    delay(3000);
    lcd.clear();
   }
  } else {
   lcd.setCursor(0, 0);
   lcd.print("OK, no changes");
   delay(1500);
  }
  lcd.setCursor(0, 0);
  lcd.print("                ");
 }
}
```