```
1: /* Copyright (C) Jelle Spijker - All Rights Reserved
2:  * Unauthorized copying of this file, via any medium is strictly prohibited
3:  * and only allowed with the written consent of the author (Jelle Spijker)
4:  * This software is proprietary and confidential
5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6:  */
7:
8: #pragma once
9:
10: #include "ADC.h"
11: #include "EC12P.h"
12: #include "eqep.h"
13: #include "GPIO.h"
14: #include "PWM.h"
15: #include "SoilCape.h"
16: #include "Microscope.h"
17: #include "CouldNotGrabImageException.h"
18: #include "ADCReadException.h"
19: #include "FailedToCreateGPIOPollingThreadException.h"
20: #include "FailedToCreateThreadException.h"
21: #include "GPIOReadException.h"
22: #include "MicroscopeNotFoundException.h"
23: #include "ValueOutOfBoundsException.h"
```

```
1: /* Copyright (C) Jelle Spijker - All Rights Reserved
2:  * Unauthorized copying of this file, via any medium is strictly prohibited
3:  * and only allowed with the written consent of the author (Jelle Spijker)
4:  * This software is proprietary and confidential
5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6:  */
7:
8: #pragma once
9:
10: #include <stdio.h>
11: #include <unistd.h>
12: #include <fcntl.h>
13: #include <errno.h>
14: #include <sys/ioctl.h>
15:
16: #include <linux/usbdevice_fs.h>
17:
18: namespace Hardware {
19: class USB {
20: public:
21:   USB();
22:   ~USB();
23:   void ResetUSB();
24: };
25: }
```

```
1: /* Copyright (C) Jelle Spijker - All Rights Reserved
2:  * Unauthorized copying of this file, via any medium is strictly prohibited
3:  * and only allowed with the written consent of the author (Jelle Spijker)
4:  * This software is proprietary and confidential
5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6:  */
7:
8: #pragma once
9:
10: #include <exception>
11: #include <string>
12:
13: using namespace std;
14:
15: namespace Hardware {
16: namespace Exception {
17: class GPIOReadException : public std::exception {
18: public:
19:   GPIOReadException(string m = "Can't read GPIO data!") : msg(m){};
20:   ~GPIOReadException() _GLIBCXX_USE_NOEXCEPT{};
21:   const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };
22:
23: private:
24:   string msg;
25: };
26: }
27: }
```

```
 1: /* Copyright (C) Jelle Spijker - All Rights Reserved
 2:  * Unauthorized copying of this file, via any medium is strictly prohibited
 3:  * and only allowed with the written consent of the author (Jelle Spijker)
 4:  * This software is proprietary and confidential
 5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 6:  */
 7:
 8: /*! \class BBB
 9: The core BeagleBone Black class used for all hardware related classes.
10: Consisting of universal used method, functions and variables. File operations,
11: polling and threading
12: */
13:
14: #pragma once
15:
16: #define SLOTS                                                                  \
17:   "/sys/devices/bone_capemgr.9/slots" /*!< Beaglebone capemanager slots file*/
18:
19: #include <fstream>
20: #include <sstream>
21: #include <string>
22: #include <sys/stat.h>
23: #include <pthread.h>
24: #include <unistd.h>
25: #include <sys/epoll.h>
26: #include <fcntl.h>
27: #include <regex>
28: #include <stdexcept>
29:
30: #include "GPIOReadException.h"
31: #include "FailedToCreateGPIOPollingThreadException.h"
32: #include "ValueOutOfBoundsException.h"
33:
34: using namespace std;
35:
36: namespace Hardware {
37: typedef int (*CallbackType)(
38:     int); /*!< CallbackType used to pass a function to a thread*/
39:
40: class BBB {
41: public:
42:   int debounceTime; /*!< debounce time for a button in milliseconds*/
43:
44:   BBB();
45:   ~BBB();
46:
47: protected:
48:   bool threadRunning;            /*!< used to stop the thread*/
49:   pthread_t thread;              /*!< The thread*/
50:   CallbackType callbackFunction; /*!< the callbakcfunction*/
51:
52:   bool DirectoryExist(const string &path);
53:   bool CapeLoaded(const string &shield);
54:
55:   string Read(const string &path);
56:   void Write(const string &path, const string &value);
57:
58:   /*! Converts a number to a string
59:   \param Number as typename
60:   \returns the number as a string
61:   */
62:   template <typename T> string NumberToString(T Number) {
63:     ostringstream ss;
64:     ss << Number;
65:     return ss.str();
66:   };
67:
68:   /*! Converts a string to a number
69:   \param Text the string that needs to be converted
70:   \return the number as typename
71:   */
72:   template <typename T> T StringToNumber(string Text) {
73:     stringstream ss(Text);
74:     T result;
75:     return ss >> result ? result : 0;
76:   };
77: };
78: }
```

```
  1: /* Copyright (C) Jelle Spijker - All Rights Reserved
  2:  * Unauthorized copying of this file, via any medium is strictly prohibited
  3:  * and only allowed with the written consent of the author (Jelle Spijker)
  4:  * This software is proprietary and confidential
  5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
  6:  */
  7:
  8: /*! \class EC12P
  9: Interaction with the sparksfun RGB encoder
 10: */
 11:
 12: #pragma once
 13:
 14: #include "eqep.h"
 15: #include "GPIO.h"
 16: #include "FailedToCreateThreadException.h"
 17:
 18: #include <pthread.h>
 19:
 20: using namespace std;
 21:
 22: namespace Hardware {
 23: class EC12P {
 24: public:
 25:   EC12P();
 26:   ~EC12P();
 27:
 28:   /*! Enumerator indicating the color of the encoder shaft*/
 29:   enum Color {
 30:     Red,     /*!< Red*/
 31:     Pink,    /*!< Pink*/
 32:     Blue,    /*!< Blue*/
 33:     SkyBlue, /*!< SkyBlue*/
 34:     Green,   /*!< Green*/
 35:     Yellow,  /*!< Yellow*/
 36:     White,   /*!< White*/
 37:     None     /*!< Off*/
 38:   };
 39:
 40:   void SetPixelColor(Color value);
 41:   Color GetPixelColor() { return PixelColor; };
 42:
 43:   void RainbowLoop(int sleepperiod);
 44:   void StopRainbowLoop() { threadRunning = false; };
 45:
 46:   eQEP Rotary{eQEP2, eQEP::eQEP_Mode_Absolute}; /*!< The encoder*/
 47:   GPIO Button{68};                              /*!< The pushbutton*/
 48:
 49: private:
 50:   Color PixelColor; /*!< Current shaft color*/
 51:
 52:   GPIO R{31}; /*!< Red LED*/
 53:   GPIO B{48}; /*!< Blue LED*/
 54:   GPIO G{51}; /*!< Green LED*/
 55:
 56:   pthread_t thread;   /*!< the thread*/
 57:   bool threadRunning; /*!< Bool used to stop the thread*/
 58:   int sleepperiod;    /*!< Sleep period*/
 59:   friend void *colorLoop(void *value);
 60: };
 61: void *colorLoop(void *value);
 62: }
```

```
  1: /* Copyright (C) Jelle Spijker - All Rights Reserved
  2:  * Unauthorized copying of this file, via any medium is strictly prohibited
  3:  * and only allowed with the written consent of the author (Jelle Spijker)
  4:  * This software is proprietary and confidential
  5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
  6:  */
  7:
  8: #include "EC12P.h"
  9:
 10: namespace Hardware {
 11: /*! Constructor*/
 12: EC12P::EC12P() {
 13:   // Init Rotary button
 14:   Button.SetDirection(GPIO::Input);
 15:   Button.SetEdge(GPIO::Rising);
 16:
 17:   // Init Encoder
 18:   Rotary.set_period(100000000L);
 19:
 20:   // Init Encoder color
 21:   R.SetDirection(GPIO::Output);
 22:   B.SetDirection(GPIO::Output);
 23:   G.SetDirection(GPIO::Output);
 24:   SetPixelColor(None);
 25:
 26:   threadRunning = false;
 27: }
 28:
 29: /*! De-constructor*/
 30: EC12P::~EC12P() {}
 31:
 32: /*! Set the shaft color
 33: \param value as Color enumerator
 34: */
 35: void EC12P::SetPixelColor(Color value) {
 36:   switch (value) {
 37:   case Hardware::EC12P::Red:
 38:     R.SetValue(GPIO::High);
 39:     B.SetValue(GPIO::Low);
 40:     G.SetValue(GPIO::Low);
 41:     break;
 42:   case Hardware::EC12P::Pink:
 43:     R.SetValue(GPIO::High);
 44:     B.SetValue(GPIO::High);
 45:     G.SetValue(GPIO::Low);
 46:     break;
 47:   case Hardware::EC12P::Blue:
 48:     R.SetValue(GPIO::Low);
 49:     B.SetValue(GPIO::High);
 50:     G.SetValue(GPIO::Low);
 51:     break;
 52:   case Hardware::EC12P::SkyBlue:
 53:     R.SetValue(GPIO::Low);
 54:     B.SetValue(GPIO::High);
 55:     G.SetValue(GPIO::High);
 56:     break;
 57:   case Hardware::EC12P::Green:
 58:     R.SetValue(GPIO::Low);
 59:     B.SetValue(GPIO::Low);
 60:     G.SetValue(GPIO::High);
 61:     break;
 62:   case Hardware::EC12P::Yellow:
 63:     R.SetValue(GPIO::High);
 64:     B.SetValue(GPIO::Low);
 65:     G.SetValue(GPIO::High);
 66:     break;
 67:   case Hardware::EC12P::White:
 68:     R.SetValue(GPIO::High);
 69:     B.SetValue(GPIO::High);
 70:     G.SetValue(GPIO::High);
 71:     break;
 72:   case Hardware::EC12P::None:
 73:     R.SetValue(GPIO::Low);
 74:     B.SetValue(GPIO::Low);
 75:     G.SetValue(GPIO::Low);
 76:     break;
 77:   }
 78:   PixelColor = value;
 79: }
 80:
 81: /*! Loops through all the colors except of as a thread  */
 82: void EC12P::RainbowLoop(int sleepperiod) {
 83:   this->sleepperiod = sleepperiod;
```

```
  84:   this->threadRunning = true;
  85:   if (pthread_create(&thread, NULL, colorLoop, this)) {
  86:     throw Exception::FailedToCreateThreadException();
  87:   }
  88: }
  89:
  90: /*! The thread function that runs trough all the colors*/
  91: void *colorLoop(void *value) {
  92:   int i = 0;
  93:   EC12P *ec12p = static_cast<EC12P *>(value);
  94:   EC12P::Color pcolor;
  95:   while (ec12p->threadRunning) {
  96:     pcolor = static_cast<EC12P::Color>(i);
  97:     ec12p->SetPixelColor(pcolor);
  98:     usleep(ec12p->sleepperiod);
  99:     i++;
 100:     if (i == 6) {
 101:       i = 0;
 102:     }
 103:   }
 104:   return ec12p;
 105: }
 106: }
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once

#include <exception>
#include <string>

using namespace std;

namespace Hardware {
namespace Exception {
class FailedToCreateGPIOPollingThreadException : public std::exception {
public:
  FailedToCreateGPIOPollingThreadException(
      string m = "Failed to create GPIO polling thread!")
      : msg(m){};
  ~FailedToCreateGPIOPollingThreadException() _GLIBCXX_USE_NOEXCEPT{};
  const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };

private:
  string msg;
};
}
}
```

```cpp
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "USB.h"

namespace Hardware {
USB::USB() {}

USB::~USB() {}

void USB::ResetUSB() {
  int fd, rc;

  fd = open("/dev/bus/usb/001/002", O_WRONLY);
  rc = ioctl(fd, USBDEVFS_RESET, 0);
  if (rc < 0) {
    throw - 1;
  }
  close(fd);
}
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "GPIO.h"

namespace Hardware {
GPIO::GPIO(int number) {

  this->number = number;
  gpiopath = GPIOS + NumberToString<int>(number);

  if (!isExported(number, direction, edge)) {
    ExportPin(number);
    direction = ReadsDirection(gpiopath);
    edge = ReadsEdge(gpiopath);
  }
  usleep(250000);
}

GPIO::~GPIO() { UnexportPin(number); }

int GPIO::WaitForEdge(CallbackType callback) {
  threadRunning = true;
  callbackFunction = callback;
  if (pthread_create(&this->thread, NULL, &threadedPollGPIO,
                     static_cast<void *>(this))) {
    threadRunning = false;
    throw Exception::FailedToCreateGPIOPollingThreadException();
  }
  return 0;
}

int GPIO::WaitForEdge() {
  if (direction == Output) {
    SetDirection(Input);
  }
  int fd, i, epollfd, count = 0;
  struct epoll_event ev;
  epollfd = epoll_create(1);
  if (epollfd == -1) {
    throw Exception::FailedToCreateGPIOPollingThreadException(
        "GPIO: Failed to create epollfd!");
  }
  if ((fd = open((gpiopath + VALUE).c_str(), O_RDONLY | O_NONBLOCK)) == -1) {
    throw Exception::GPIOReadException();
  }

  // read operation | edge triggered | urgent data
  ev.events = EPOLLIN | EPOLLET | EPOLLPRI;
  ev.data.fd = fd;

  if (epoll_ctl(epollfd, EPOLL_CTL_ADD, fd, &ev) == -1) {
    throw Exception::FailedToCreateGPIOPollingThreadException(
        "GPIO: Failed to add control interface!");
  }

  while (count <= 1) {
    i = epoll_wait(epollfd, &ev, 1, -1);
    if (i == -1) {
      close(fd);
      return -1;
    } else {
      count++;
    }
  }
  close(fd);
  return 0;
}

GPIO::Value GPIO::GetValue() { return ReadsValue(gpiopath); }
void GPIO::SetValue(GPIO::Value value) { WritesValue(gpiopath, value); }

GPIO::Direction GPIO::GetDirection() { return direction; }
void GPIO::SetDirection(Direction direction) {
  this->direction = direction;
  WritesDirection(gpiopath, direction);
}

GPIO::Edge GPIO::GetEdge() { return edge; }
```

```
  84: void GPIO::SetEdge(Edge edge) {
  85:   this->edge = edge;
  86:   WritesEdge(gpiopath, edge);
  87: }
  88:
  89: bool GPIO::isExported(int number __attribute__((unused)), Direction &dir, Edge &edge) {
  90:   // Checks if directory exist and therefore is exported
  91:   if (!DirectoryExist(gpiopath)) {
  92:     return false;
  93:   }
  94:
  95:   // Reads the data associated with the pin
  96:   dir = ReadsDirection(gpiopath);
  97:   edge = ReadsEdge(gpiopath);
  98:   return true;
  99: }
 100:
 101: bool GPIO::ExportPin(int number) {
 102:   Write(EXPORT_PIN, NumberToString<int>(number));
 103:   usleep(250000);
 104: }
 105:
 106: bool GPIO::UnexportPin(int number) {
 107:   Write(UNEXPORT_PIN, NumberToString<int>(number));
 108: }
 109:
 110: GPIO::Direction GPIO::ReadsDirection(const string &gpiopath) {
 111:   if (Read(gpiopath + DIRECTION) == "in") {
 112:     return Input;
 113:   } else {
 114:     return Output;
 115:   }
 116: }
 117:
 118: void GPIO::WritesDirection(const string &gpiopath, Direction direction) {
 119:   switch (direction) {
 120:   case Hardware::GPIO::Input:
 121:     Write((gpiopath + DIRECTION), "in");
 122:     break;
 123:   case Hardware::GPIO::Output:
 124:     Write((gpiopath + DIRECTION), "out");
 125:     break;
 126:   }
 127: }
 128:
 129: GPIO::Edge GPIO::ReadsEdge(const string &gpiopath) {
 130:   string reader = Read(gpiopath + EDGE);
 131:   if (reader == "none") {
 132:     return None;
 133:   } else if (reader == "rising") {
 134:     return Rising;
 135:   } else if (reader == "falling") {
 136:     return Falling;
 137:   } else {
 138:     return Both;
 139:   }
 140: }
 141:
 142: void GPIO::WritesEdge(const string &gpiopath, Edge edge) {
 143:   switch (edge) {
 144:   case Hardware::GPIO::None:
 145:     Write((gpiopath + EDGE), "none");
 146:     break;
 147:   case Hardware::GPIO::Rising:
 148:     Write((gpiopath + EDGE), "rising");
 149:     break;
 150:   case Hardware::GPIO::Falling:
 151:     Write((gpiopath + EDGE), "falling");
 152:     break;
 153:   case Hardware::GPIO::Both:
 154:     Write((gpiopath + EDGE), "both");
 155:     break;
 156:   default:
 157:     break;
 158:   }
 159: }
 160:
 161: GPIO::Value GPIO::ReadsValue(const string &gpiopath) {
 162:   string path(gpiopath + VALUE);
 163:   int res = StringToNumber<int>(Read(path));
 164:   return (Value)res;
 165: }
 166:
```

```
167: void GPIO::WritesValue(const string &gpiopath, Value value) {
168:   Write(gpiopath + VALUE, NumberToString<int>(value));
169: }
170:
171: void *threadedPollGPIO(void *value) {
172:   GPIO *gpio = static_cast<GPIO *>(value);
173:   while (gpio->threadRunning) {
174:     gpio->callbackFunction(gpio->WaitForEdge());
175:     usleep(gpio->debounceTime * 1000);
176:   }
177:   return 0;
178: }
179: }
```

```
 1: /* Copyright (C) Jelle Spijker - All Rights Reserved
 2:  * Unauthorized copying of this file, via any medium is strictly prohibited
 3:  * and only allowed with the written consent of the author (Jelle Spijker)
 4:  * This software is proprietary and confidential
 5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 6:  */
 7:
 8: /*! \class Microscope
 9: Interaction with the USB 5 MP microscope
10: */
11:
12: #pragma once
13: #define MICROSCOPE_VERSION 1 /*!< Version of the class*/
14:
15: #define MICROSCOPE_NAME "USB Microscope"
16: #define MIN_BRIGHTNESS -64
17: #define MAX_BRIGHTNESS 64
18: #define MIN_CONTRAST 0
19: #define MAX_CONTRAST 64
20: #define MIN_SATURATION 0
21: #define MAX_SATURATION 128
22: #define MIN_HUE -40
23: #define MAX_HUE 40
24: #define MIN_GAMMA 40
25: #define MAX_GAMMA 500
26: #define MIN_SHARPNESS 1
27: #define MAX_SHARPNESS 25
28:
29: #include "stdint.h"
30: #include <vector>
31: #include <string>
32: #include <sys/stat.h>
33: #include <sys/utsname.h>
34:
35: #include <boost/signals2.hpp>
36: #include <boost/bind.hpp>
37:
38: #include "USB.h"
39:
40: #include <opencv2/photo.hpp>
41: #include <opencv2/imgcodecs.hpp>
42: #include <opencv2/opencv.hpp>
43: #include <opencv/highgui.h>
44: #include <opencv2/videoio.hpp>
45:
46: #include <boost/filesystem.hpp>
47:
48: #include <fstream>
49:
50: namespace Hardware {
51: class Microscope {
52: public:
53:   /*! Struct that represent the Resolution that is used */
54:   struct Resolution {
55:   public:
56:     uint16_t Width;  /*!< Width of the image*/
57:     uint16_t Height; /*!< Height of the image*/
58:   };
59:
60:   typedef boost::signals2::signal<void()> Finished_t;
61:   typedef boost::signals2::signal<void(int)> Progress_t;
62:
63:   boost::signals2::connection
64:   connect_Finished(const Finished_t::slot_type &subscriber);
65:   boost::signals2::connection
66:   connect_Progress(const Progress_t::slot_type &subscriber);
67:
68:   uint8_t FrameDelayTrigger; /*!< Delay in seconds */
69:   cv::Mat LastFrame;         /*!< Last grabbed and processed frame */
70:   Resolution Dimensions;     /*!< Dimensions of the frame */
71:
72:   Microscope(uint8_t frameDelayTrigger = 3,
73:              Resolution dimensions = Resolution{2048, 1536},
74:              bool firstdefault = true);
75:   ~Microscope();
76:
77:   static std::vector<std::string> AvailableCams();
78:
79:   void GetFrame(cv::Mat &dst);
80:   void GetHDRFrame(cv::Mat &dst, uint32_t noframes = 5);
81:
82:   bool IsOpened();
83:   void Release();
```

```
  84:
  85:   void openCam(int dev);
  86:
  87: private:
  88:   Finished_t fin_sig;
  89:   Progress_t prog_sig;
  90:
  91:   std::string arch;
  92:
  93:   cv::VideoCapture
  94:       captureDevice; /*!< An openCV instance of the capture device*/
  95:   void StartupSeq(bool firstdefault);
  96:
  97:   std::vector<cv::Mat> HDRframes;
  98:   std::vector<float> times;
  99:
 100:   static bool exist(const std::string &name);
 101: };
 102: }
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once
#include <exception>
#include <string>

using namespace std;

namespace Hardware {
namespace Exception {
class MicroscopeNotFoundException : public std::exception {
public:
  MicroscopeNotFoundException(string m = "Microscope not found exception!")
      : msg(m){};
  ~MicroscopeNotFoundException() _GLIBCXX_USE_NOEXCEPT{};
  const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };

private:
  string msg;
};
}
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "Microscope.h"

// Custom exceptions
#include "MicroscopeNotFoundException.h"
#include "CouldNotGrabImageException.h"

using namespace cv;
using namespace boost::filesystem;
using namespace std;

namespace Hardware {

/*! Constructor of the class which initializes the USB microscope
\param frameDelayTrigger the delay between the first initialization of the
microscope and the retrivial of the image expressed in seconds. Default value is
3 seconds
\param dimension A resolution Struct indicating which resolution the webcam
should use. Default is 2592 x 1944
*/
Microscope::Microscope(uint8_t frameDelayTrigger, Resolution dimensions,
                       bool firstdefault) {
  FrameDelayTrigger = frameDelayTrigger;
  Dimensions = dimensions;

  StartupSeq(firstdefault);
}

void Microscope::StartupSeq(bool firstdefault) {
  std::vector<std::string> camNames = AvailableCams();
  uint videodev = find(camNames.begin(), camNames.end(), MICROSCOPE_NAME) -
                  camNames.begin();
  if (videodev == camNames.size() && !firstdefault) {
    throw Exception::MicroscopeNotFoundException(
        "Microscope Not Found Exception!");
  } else if (videodev == camNames.size() && firstdefault) {
    videodev = 0;
  }

  struct utsname unameData;
  uname(&unameData);
  arch = static_cast<std::string>(unameData.machine);

  try {
    openCam(videodev);
  } catch (Exception::MicroscopeNotFoundException &e) {
    // Tries to soft reset the USB port. Haven't got this working yet
    USB usbdev;
    usbdev.ResetUSB();
    captureDevice.open(videodev);
    if (!captureDevice.isOpened()) {
      throw Exception::MicroscopeNotFoundException("USB Soft Reset Exception!");
    }
  }
}

/*!< De-constructor*/
Microscope::~Microscope() { captureDevice.~VideoCapture(); }

/*! Get the frame after the set initialization period
\param dst a cv::Mat construct which stores the retrieved image
*/
void Microscope::GetFrame(cv::Mat &dst) {
  // Work around for crappy cam retrival of the BBB
  if (arch.find("armv7l") != string::npos) {
    if (!captureDevice.grab()) {
      throw Exception::CouldNotGrabImageException();
    }
    sleep(FrameDelayTrigger); // Needed otherwise scrambled picture
    if (!captureDevice.grab()) {
      throw Exception::CouldNotGrabImageException();
    }
    captureDevice.retrieve(dst);
  } else {
    if (!captureDevice.read(dst)) {
      throw Exception::CouldNotGrabImageException();
    }
```

```
  }
}

/*! Get an HDR capture of the cam using a user defined number of frames
differently lit frames. Due to hardware limitations each frames take roughly 3
seconds to grab. This function is based upon the tutorial from openCV
http://docs.opencv.org/trunk/doc/tutorials/photo/hdr_imaging/hdr_imaging.html
\param dst a cv::Mat construct with the retrieved HDR result
\param noframes is the number of frames that create the HDR image - default = 5
*/
void Microscope::GetHDRFrame(cv::Mat &dst, uint32_t noframes) {
  prog_sig(0);
  // create the brightness steps
  int8_t brightnessStep =
      static_cast<int8_t>((MAX_BRIGHTNESS - MIN_BRIGHTNESS) / noframes);
  int8_t currentBrightness = captureDevice.get(CV_CAP_PROP_BRIGHTNESS);
  int8_t currentContrast = captureDevice.get(CV_CAP_PROP_CONTRAST);
  captureDevice.set(CV_CAP_PROP_CONTRAST, MAX_CONTRAST);

  int progStep = 70 / noframes;
  Mat currentImg;
  // take the shots at different brightness levels
  for (uint32_t i = 1; i <= noframes; i++) {
    captureDevice.set(CV_CAP_PROP_BRIGHTNESS,
                      (MIN_BRIGHTNESS + (i * brightnessStep)));
    GetFrame(currentImg);
    HDRframes.push_back(currentImg);
    prog_sig(i * progStep);
  }

  // Set the brightness and back to the previous used level
  captureDevice.set(CV_CAP_PROP_BRIGHTNESS, currentBrightness);
  captureDevice.set(CV_CAP_PROP_CONTRAST, currentContrast);

  // Perform the exposure fusion
  Mat fusion;
  Ptr<MergeMertens> merge_mertens = createMergeMertens();
  merge_mertens->process(HDRframes, fusion);
  prog_sig(80);
  fusion *= 255;
  prog_sig(85);
  fusion.convertTo(dst, CV_8UC1);
  prog_sig(100);
  fin_sig();
}

/*!< Checks if the capture device is open and returns the status as a bool
/return Status of the capture device expressed as a bool
*/
bool Microscope::IsOpened() { return captureDevice.isOpened(); }

/*!< Safely release the capture device*/
void Microscope::Release() { captureDevice.release(); }

/*!< Opens the webcam*/
void Microscope::openCam(int dev) {
  captureDevice.open(dev);
  if (!captureDevice.isOpened()) {
    throw Exception::MicroscopeNotFoundException();
  }
  captureDevice.set(CV_CAP_PROP_FRAME_WIDTH, Dimensions.Width);
  captureDevice.set(CV_CAP_PROP_FRAME_HEIGHT, Dimensions.Height);
}

std::vector<std::string> Microscope::AvailableCams() {
  std::vector<std::string> cams;
  const string path_ss = "/sys/class/video4linux";

  if (!exist(path_ss))
    return cams;

  for (directory_iterator itr(path_ss); itr != directory_iterator(); ++itr) {
    string videoln = itr->path().string();
    videoln.append("/name");
    if (exist(videoln)) {
      std::ifstream camName;
      camName.open(videoln);
      std::string name;
      std::getline(camName, name);
      cams.push_back(name);
      camName.close();
    }
  }
```

```
167:
168:   return cams;
169: }
170:
171: bool Microscope::exist(const string &name) {
172:   struct stat buffer;
173:   return (stat(name.c_str(), &buffer) == 0);
174: }
175:
176: boost::signals2::connection
177: Microscope::connect_Finished(const Finished_t::slot_type &subscriber) {
178:   return fin_sig.connect(subscriber);
179: }
180:
181: boost::signals2::connection
182: Microscope::connect_Progress(const Progress_t::slot_type &subscriber) {
183:   return prog_sig.connect(subscriber);
184: }
185: }
```

```
 1: /* Copyright (C) Jelle Spijker - All Rights Reserved
 2:  * Unauthorized copying of this file, via any medium is strictly prohibited
 3:  * and only allowed with the written consent of the author (Jelle Spijker)
 4:  * This software is proprietary and confidential
 5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 6:  */
 7:
 8: #pragma once
 9: #include <exception>
10: #include <string>
11:
12: using namespace std;
13:
14: namespace Hardware {
15: namespace Exception {
16: class ADCReadException : public std::exception {
17: public:
18:   ADCReadException(string m = "Can't read ADC data!") : msg(m){};
19:   ~ADCReadException() _GLIBCXX_USE_NOEXCEPT{};
20:   const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };
21:
22: private:
23:   string msg;
24: };
25: }
26: }
```

```
1: /* Copyright (C) Jelle Spijker - All Rights Reserved
2:  * Unauthorized copying of this file, via any medium is strictly prohibited
3:  * and only allowed with the written consent of the author (Jelle Spijker)
4:  * This software is proprietary and confidential
5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6:  */
7:
8: #pragma once
9:
10: #include "EC12P.h"
11: #include "GPIO.h"
12: #include "PWM.h"
13: #include "ADC.h"
14:
15: namespace Hardware {
16: class SoilCape {
17: public:
18:   EC12P RGBEncoder;
19:   PWM MicroscopeLEDs{PWM::P9_14};
20:   ADC MicroscopeLDR{ADC::ADC0};
21:
22:   SoilCape();
23:   ~SoilCape();
24: };
25: }
```

```
 1: /* Copyright (C) Jelle Spijker - All Rights Reserved
 2:  * Unauthorized copying of this file, via any medium is strictly prohibited
 3:  * and only allowed with the written consent of the author (Jelle Spijker)
 4:  * This software is proprietary and confidential
 5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 6:  */
 7:
 8: #pragma once
 9: #include <exception>
10: #include <string>
11:
12: using namespace std;
13:
14: namespace Hardware {
15: namespace Exception {
16: class CouldNotGrabImageException : public std::exception {
17: public:
18:   CouldNotGrabImageException(string m = "Unable to grab the next image!")
19:       : msg(m){};
20:   ~CouldNotGrabImageException() _GLIBCXX_USE_NOEXCEPT{};
21:   const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };
22:
23: private:
24:   string msg;
25: };
26: }
27: }
```

```
  1: /*
  2: * TI eQEP driver interface API
  3: *
  4: * Copyright (C) 2013 Nathaniel R. Lewis - http://nathanielrlewis.com/
  5: *
  6: * This program is free software; you can redistribute it and/or modify
  7: * it under the terms of the GNU General Public License as published by
  8: * the Free Software Foundation; either version 2 of the License, or
  9: * (at your option) any later version.
 10: *
 11: * This program is distributed in the hope that it will be useful,
 12: * but WITHOUT ANY WARRANTY; without even the implied warranty of
 13: * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 14: * GNU General Public License for more details.
 15: *
 16: * You should have received a copy of the GNU General Public License
 17: * along with this program; if not, write to the Free Software
 18: * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 19: *
 20: *
 21: * This code is changed by Jelle Spijker (C) 2014.
 22: * Introducing polling with threading.
 23: *
 24: */
 25:
 26: #pragma once
 27:
 28: #include <iostream>
 29: #include <stdint.h>
 30: #include <string>
 31: #include "BBB.h"
 32:
 33: #define eQEP0 "/sys/devices/ocp.3/48300000.epwmss/48300180.eqep"
 34: #define eQEP1 "/sys/devices/ocp.3/48302000.epwmss/48302180.eqep"
 35: #define eQEP2 "/sys/devices/ocp.3/48304000.epwmss/48304180.eqep"
 36:
 37: namespace Hardware {
 38: // Class which defines an interface to my eQEP driver
 39: class eQEP : public BBB {
 40:   // Base path for the eQEP unit
 41:   std::string path;
 42:
 43: public:
 44:   // Modes of operation for the eQEP hardware
 45:   typedef enum {
 46:     // Absolute positioning mode
 47:     eQEP_Mode_Absolute = 0,
 48:
 49:     // Relative positioning mode
 50:     eQEP_Mode_Relative = 1,
 51:
 52:     // Error flag
 53:     eQEP_Mode_Error = 2,
 54:   } eQEP_Mode;
 55:
 56:   // Default constructor for the eQEP interface driver
 57:   eQEP(std::string _path, eQEP_Mode _mode);
 58:
 59:   // Reset the value of the encoder
 60:   void set_position(int32_t position);
 61:
 62:   // Get the position of the encoder, pass poll as true to poll the pin, whereas
 63:   // passing false reads the immediate value
 64:   int32_t get_position(bool _poll = true);
 65:
 66:   // Thread of the poll
 67:   int WaitForPositionChange(CallbackType callback);
 68:   void WaitForPositionChangeCancel() { this->threadRunning = false; }
 69:
 70:   // Set the polling period
 71:   void set_period(long long unsigned int period);
 72:
 73:   // Get the polling period of the encoder
 74:   uint64_t get_period();
 75:
 76:   // Set the mode of the eQEP hardware
 77:   void set_mode(eQEP_Mode mode);
 78:
 79:   // Get the mode of the eQEP hardware
 80:   eQEP_Mode get_mode();
 81:
 82: private:
 83:   friend void *threadedPolleqep(void *value);
```

```
84: };
85:
86: void *threadedPolleqep(void *value);
87: }
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#pragma once
#include "BBB.h"
#include <dirent.h>

#define OCP_PATH "/sys/devices/ocp.3/"
#define P8_13_FIND "bs_pwm_test_P8_13"
#define P8_19_FIND "bs_pwm_test_P8_19"
#define P9_14_FIND "bs_pwm_test_P9_14"
#define P9_16_FIND "bs_pwm_test_P9_16"

#define PWM_CAPE "am33xx_pwm"
#define P8_13_CAPE "bspwm_P8_13" //_14
#define P8_19_CAPE "bspwm_P8_13" //_14
#define P9_14_CAPE "bspwm_P9_14" //_16
#define P9_16_CAPE "bspwm_P9_16" //_16

#define P8_13_CAPE_LOAD "bspwm_P8_13_14"
#define P8_19_CAPE_LOAD "bspwm_P8_13_14"
#define P9_14_CAPE_LOAD "bspwm_P9_14_16"
#define P9_16_CAPE_LOAD "bspwm_P9_16_16"

namespace Hardware {
class PWM : public BBB {
public:
  enum Pin // Four possible PWM pins
  { P8_13,
    P8_19,
    P9_14,
    P9_16 };
  enum Run // Signal generating
  { On = 1,
    Off = 0 };
  enum Polarity // Inverse duty polarity
  { Normal = 1,
    Inverted = 0 };

  Pin pin; // Current pin

  uint8_t GetPixelValue() { return pixelvalue; }
  void SetPixelValue(uint8_t value);

  float GetIntensity() { return intensity; };
  void SetIntensity(float value);

  int GetPeriod() { return period; };
  void SetPeriod(int value);

  int GetDuty() { return duty; };
  void SetDuty(int value);
  void SetIntensity();

  Run GetRun() { return run; };
  void SetRun(Run value);

  Polarity GetPolarity() { return polarity; };
  void SetPolarity(Polarity value);

  PWM(Pin pin);
  ~PWM();

private:
  int period;         // current period
  int duty;           // current duty
  float intensity;    // current intensity
  uint8_t pixelvalue; // current pixelvalue
  Run run;            // current run state
  Polarity polarity;  // current polaity

  string basepath;     // the basepath ocp.3
  string dutypath;     // base + duty path
  string periodpath;   // base + period path
  string runpath;      // base + run path
  string polaritypath; // base + polarity path

  void calcIntensity();
  string FindPath(string value);
```

```
84: };
85: }
```

```cpp
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "SoilCape.h"

namespace Hardware {
SoilCape::SoilCape() {}

SoilCape::~SoilCape() {}
}
```

```
1: /* Copyright (C) Jelle Spijker - All Rights Reserved
2:  * Unauthorized copying of this file, via any medium is strictly prohibited
3:  * and only allowed with the written consent of the author (Jelle Spijker)
4:  * This software is proprietary and confidential
5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6:  */
7:
8: #pragma once
9:
10: #include <exception>
11: #include <string>
12:
13: using namespace std;
14:
15: namespace Hardware {
16: namespace Exception {
17: class ValueOutOfBoundsException : public std::exception {
18: public:
19:   ValueOutOfBoundsException(string m = "Value out of bounds!") : msg(m){};
20:   ~ValueOutOfBoundsException() _GLIBCXX_USE_NOEXCEPT{};
21:   const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };
22:
23: private:
24:   string msg;
25: };
26: }
27: }
```

```cpp
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "ADC.h"

namespace Hardware {
/*! Constructor
\param pin and ADCPin type indicating which analogue pin to use
*/
ADC::ADC(ADCPin pin) {
  this->Pin = pin;
  switch (pin) {
  case Hardware::ADC::ADC0:
    adcpath = ADC0_PATH;
    break;
  case Hardware::ADC::ADC1:
    adcpath = ADC1_PATH;
    break;
  case Hardware::ADC::ADC2:
    adcpath = ADC2_PATH;
    break;
  case Hardware::ADC::ADC3:
    adcpath = ADC3_PATH;
    break;
  case Hardware::ADC::ADC4:
    adcpath = ADC4_PATH;
    break;
  case Hardware::ADC::ADC5:
    adcpath = ADC5_PATH;
    break;
  case Hardware::ADC::ADC6:
    adcpath = ADC6_PATH;
    break;
  case Hardware::ADC::ADC7:
    adcpath = ADC7_PATH;
    break;
  }

  MinIntensity = 0;
  MaxIntensity = 4096;
}

/*! De-constructor*/
ADC::~ADC() {}

/*! Reads the current voltage in the pin
\return an integer between 0 and 4096
*/
int ADC::GetCurrentValue() {
  int retVal = StringToNumber<int>(Read(adcpath));
  Intensity = (float)(retVal - MinIntensity) /
              (4096 - (MinIntensity + (4096 - MaxIntensity)));
  return retVal;
}

/*! Set the current voltage at the pin as the minimum voltage*/
void ADC::SetMinIntensity() {
  MinIntensity = StringToNumber<int>(Read(adcpath));
}

void ADC::SetMaxIntensity() {
  MaxIntensity = StringToNumber<int>(Read(adcpath));
}

/*! Threading enabled polling of the analogue pin
\param callback the function which should be called when polling indicates a
change CallbackType
\return 0
*/
int ADC::WaitForValueChange(CallbackType callback) {
  threadRunning = true;
  callbackFunction = callback;
  if (pthread_create(&thread, NULL, &threadedPollADC,
                     static_cast<void *>(this))) {
    threadRunning = false;
    throw Exception::FailedToCreateGPIOPollingThreadException();
  }
  return 0;
}
```

```
 84:
 85: /*! Polling of the analogue pin
 86: \return the current value
 87: */
 88: int ADC::WaitForValueChange() {
 89:   int fd, i, epollfd, count = 0;
 90:   struct epoll_event ev;
 91:   epollfd = epoll_create(1);
 92:   if (epollfd == -1) {
 93:     throw Exception::FailedToCreateGPIOPollingThreadException(
 94:         "GPIO: Failed to create epollfd!");
 95:   }
 96:   if ((fd = open(adcpath.c_str(), O_RDONLY | O_NONBLOCK)) == -1) {
 97:     throw Exception::ADCReadException();
 98:   }
 99:   ev.events = EPOLLIN;
100:   ev.data.fd = fd;
101:
102:   if (epoll_ctl(epollfd, EPOLL_CTL_ADD, fd, &ev) == -1) {
103:     throw Exception::FailedToCreateGPIOPollingThreadException(
104:         "ADC: Failed to add control interface!");
105:   }
106:
107:   while (count <= 1) {
108:     i = epoll_wait(epollfd, &ev, 1, -1);
109:     if (i == -1) {
110:       close(fd);
111:       return -1;
112:     } else {
113:       count++;
114:     }
115:   }
116:   close(fd);
117:   return StringToNumber<int>(Read(adcpath));
118: }
119:
120: /*! friendly function to start the threading*/
121: void *threadedPollADC(void *value) {
122:   ADC *adc = static_cast<ADC *>(value);
123:   while (adc->threadRunning) {
124:     adc->callbackFunction(adc->WaitForValueChange());
125:     usleep(200000);
126:   }
127: }
128: }
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

/*! \class ADC
Interaction with the beaglebone analogue pins
*/

#pragma once

#include "BBB.h"
#include "ADCReadException.h"

#define ADC0_PATH                                                       \
  "/sys/bus/iio/devices/iio:device0/in_voltage0_raw" /*!< path to analogue pin \
                                                       0*/
#define ADC1_PATH                                                       \
  "/sys/bus/iio/devices/iio:device0/in_voltage1_raw" /*!< path to analogue pin \
                                                       1*/
#define ADC2_PATH                                                       \
  "/sys/bus/iio/devices/iio:device0/in_voltage2_raw" /*!< path to analogue pin \
                                                       2*/
#define ADC3_PATH                                                       \
  "/sys/bus/iio/devices/iio:device0/in_voltage3_raw" /*!< path to analogue pin \
                                                       3*/
#define ADC4_PATH                                                       \
  "/sys/bus/iio/devices/iio:device0/in_voltage4_raw" /*!< path to analogue pin \
                                                       4*/
#define ADC5_PATH                                                       \
  "/sys/bus/iio/devices/iio:device0/in_voltage5_raw" /*!< path to analogue pin \
                                                       5*/
#define ADC6_PATH                                                       \
  "/sys/bus/iio/devices/iio:device0/in_voltage6_raw" /*!< path to analogue pin \
                                                       6*/
#define ADC7_PATH                                                       \
  "/sys/bus/iio/devices/iio:device0/in_voltage7_raw" /*!< path to analogue pin \
                                                       7*/

namespace Hardware {
class ADC : public BBB {
public:
  /*! Enumerator to indicate the analogue pin*/
  enum ADCPin {
    ADC0, /*!< AIN0 pin*/
    ADC1, /*!< AIN1 pin*/
    ADC2, /*!< AIN2 pin*/
    ADC3, /*!< AIN3 pin*/
    ADC4, /*!< AIN4 pin*/
    ADC5, /*!< AIN5 pin*/
    ADC6, /*!< AIN6 pin*/
    ADC7  /*!< AIN7 pin*/
  };

  ADCPin Pin; /*!< current pin*/

  ADC(ADCPin pin);
  ~ADC();

  int GetCurrentValue();
  float GetIntensity() { return Intensity; }
  int GetMinIntensity() { return MinIntensity; }
  int GetMaxIntensity() { return MaxIntensity; }

  void SetMinIntensity();
  void SetMaxIntensity();

  int WaitForValueChange();
  int WaitForValueChange(CallbackType callback);
  void WaitForValueChangeCancel() { this->threadRunning = false; }

private:
  string adcpath;   /*!< Path to analogue write file*/
  float Intensity;  /*!< Current intensity expressed as percentage*/
  int MinIntensity; /*!< Voltage level which represent 0 percentage*/
  int MaxIntensity; /*!< Voltage level which represent 100 percentage*/

  friend void *threadedPollADC(void *value); /*!< friend polling function*/
};

void *threadedPollADC(void *value);
```

```
84: }
```

```
  1: /*
  2: * TI eQEP driver interface API
  3: *
  4: * Copyright (C) 2013 Nathaniel R. Lewis - http://nathanielrlewis.com/
  5: *
  6: * This program is free software; you can redistribute it and/or modify
  7: * it under the terms of the GNU General Public License as published by
  8: * the Free Software Foundation; either version 2 of the License, or
  9: * (at your option) any later version.
 10: *
 11: * This program is distributed in the hope that it will be useful,
 12: * but WITHOUT ANY WARRANTY; without even the implied warranty of
 13: * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 14: * GNU General Public License for more details.
 15: *
 16: * You should have received a copy of the GNU General Public License
 17: * along with this program; if not, write to the Free Software
 18: * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 19: *
 20: * This file is modified by Jelle Spijker 2014
 21: * Added polling and threading capabilties
 22: *
 23: */
 24:
 25: // Pull in our eQEP driver definitions
 26: #include "eqep.h"
 27:
 28: // Language dependencies
 29: #include <cstdint>
 30: #include <cstdlib>
 31: #include <cstdio>
 32:
 33: // POSIX dependencies
 34: #include <unistd.h>
 35: #include <fcntl.h>
 36: #include <poll.h>
 37: #include <sys/types.h>
 38: #include <sys/stat.h>
 39:
 40: namespace Hardware {
 41: // Constructor for eQEP driver interface object
 42: eQEP::eQEP(std::string _path, eQEP::eQEP_Mode _mode) : path(_path) {
 43:   if (_path == eQEP0) {
 44:     if (!CapeLoaded("bone_eqep0")) {
 45:       Write(SLOTS, "bone_eqep0");
 46:     }
 47:   } else if (_path == eQEP1) {
 48:     if (!CapeLoaded("bone_eqep1")) {
 49:       Write(SLOTS, "bone_eqep1");
 50:     }
 51:   } else if (_path == eQEP2) {
 52:     if (!CapeLoaded("bone_eqep2b")) {
 53:       Write(SLOTS, "bone_eqep2b");
 54:     }
 55:   }
 56:
 57:   // Set the mode of the hardware
 58:   this->set_mode(_mode);
 59:
 60:   // Reset the position
 61:   this->set_position(0);
 62: }
 63:
 64: // Set the position of the eQEP hardware
 65: void eQEP::set_position(int32_t position) {
 66:   // Open the file representing the position
 67:   FILE *fp = fopen((this->path + "/position").c_str(), "w");
 68:
 69:   // Check that we opened the file correctly
 70:   if (fp == NULL) {
 71:     // Error, break out
 72:     std::cerr << "[eQEP " << this->path << "] Unable to open position for write"
 73:               << std::endl;
 74:     return;
 75:   }
 76:
 77:   // Write the desired value to the file
 78:   fprintf(fp, "%d\n", position);
 79:
 80:   // Commit changes
 81:   fclose(fp);
 82: }
 83:
```

```
  84: // Set the period of the eQEP hardware
  85: void eQEP::set_period(long long unsigned int period) {
  86:   // Open the file representing the position
  87:   FILE *fp = fopen((this->path + "/period").c_str(), "w");
  88:
  89:   // Check that we opened the file correctly
  90:   if (fp == NULL) {
  91:     // Error, break out
  92:     std::cerr << "[eQEP " << this->path << "] Unable to open period for write"
  93:               << std::endl;
  94:     return;
  95:   }
  96:
  97:   // Write the desired value to the file
  98:   fprintf(fp, "%llu\n", period);
  99:
 100:   // Commit changes
 101:   fclose(fp);
 102: }
 103:
 104: // Set the mode of the eQEP hardware
 105: void eQEP::set_mode(eQEP::eQEP_Mode _mode) {
 106:   // Open the file representing the position
 107:   FILE *fp = fopen((this->path + "/mode").c_str(), "w");
 108:
 109:   // Check that we opened the file correctly
 110:   if (fp == NULL) {
 111:     // Error, break out
 112:     std::cerr << "[eQEP " << this->path << "] Unable to open mode for write"
 113:               << std::endl;
 114:     return;
 115:   }
 116:
 117:   // Write the desired value to the file
 118:   fprintf(fp, "%u\n", _mode);
 119:
 120:   // Commit changes
 121:   fclose(fp);
 122: }
 123:
 124: int eQEP::WaitForPositionChange(CallbackType callback) {
 125:   threadRunning = true;
 126:   callbackFunction = callback;
 127:   if (pthread_create(&this->thread, NULL, &threadedPolleqep,
 128:                      static_cast<void *>(this))) {
 129:     threadRunning = false;
 130:     throw Exception::FailedToCreateGPIOPollingThreadException();
 131:   }
 132:
 133:   return 0;
 134: }
 135:
 136: // Get the position of the hardware
 137: int32_t eQEP::get_position(bool _poll) {
 138:   // Position temporary variable
 139:   int32_t position;
 140:   char dummy;
 141:   struct pollfd ufd;
 142:
 143:   // Do we want to poll?
 144:   if (_poll) {
 145:     // Open a connection to the attribute file.
 146:     if ((ufd.fd = open((this->path + "/position").c_str(), O_RDWR)) < 0) {
 147:       // Error, break out
 148:       std::cerr << "[eQEP " << this->path
 149:                 << "] unable to open position for polling" << std::endl;
 150:       return 0;
 151:     }
 152:
 153:     // Dummy read
 154:     read(ufd.fd, &dummy, 1);
 155:
 156:     // Poll the port
 157:     ufd.events = (short)EPOLLET;
 158:     if (poll(&ufd, 1, -1) < 0) {
 159:       // Error, break out
 160:       std::cerr << "[eQEP " << this->path << "] Error occurred whilst polling"
 161:                 << std::endl;
 162:       close(ufd.fd);
 163:       return 0;
 164:     }
 165:   }
 166:
```

```
167:   // Read the position
168:   FILE *fp = fopen((this->path + "/position").c_str(), "r");
169:
170:   // Check that we opened the file correctly
171:   if (fp == NULL) {
172:     // Error, break out
173:     std::cerr << "[eQEP " << this->path << "] Unable to open position for read"
174:               << std::endl;
175:     close(ufd.fd);
176:     return 0;
177:   }
178:
179:   // Write the desired value to the file
180:   fscanf(fp, "%d", &position);
181:
182:   // Commit changes
183:   fclose(fp);
184:
185:   // If we were polling, close the polling file
186:   if (_poll) {
187:     close(ufd.fd);
188:   }
189:
190:   // Return the position
191:   return position;
192: }
193:
194: // Get the period of the eQEP hardware
195: uint64_t eQEP::get_period() {
196:   // Open the file representing the position
197:   FILE *fp = fopen((this->path + "/period").c_str(), "r");
198:
199:   // Check that we opened the file correctly
200:   if (fp == NULL) {
201:     // Error, break out
202:     std::cerr << "[eQEP " << this->path << "] Unable to open period for read"
203:               << std::endl;
204:     return 0;
205:   }
206:
207:   // Write the desired value to the file
208:   uint64_t period = 0;
209:   fscanf(fp, "%llu", &period);
210:
211:   // Commit changes
212:   fclose(fp);
213:
214:   // Return the period
215:   return period;
216: }
217:
218: // Get the mode of the eQEP hardware
219: eQEP::eQEP_Mode eQEP::get_mode() {
220:   // Open the file representing the position
221:   FILE *fp = fopen((this->path + "/mode").c_str(), "r");
222:
223:   // Check that we opened the file correctly
224:   if (fp == NULL) {
225:     // Error, break out
226:     std::cerr << "[eQEP " << this->path << "] Unable to open mode for read"
227:               << std::endl;
228:     return eQEP::eQEP_Mode_Error;
229:   }
230:
231:   // Write the desired value to the file
232:   eQEP::eQEP_Mode mode;
233:   fscanf(fp, "%u", (unsigned int *)&mode);
234:
235:   // Commit changes
236:   fclose(fp);
237:
238:   // Return the mode
239:   return mode;
240: }
241:
242: void *threadedPolleqep(void *value) {
243:   eQEP *eqep = static_cast<eQEP *>(value);
244:   while (eqep->threadRunning) {
245:     eqep->callbackFunction(eqep->get_position(true));
246:     usleep(eqep->debounceTime * 1000);
247:   }
248:   return 0;
249: }
```

```
250: }
```

```
1: /* Copyright (C) Jelle Spijker - All Rights Reserved
2:  * Unauthorized copying of this file, via any medium is strictly prohibited
3:  * and only allowed with the written consent of the author (Jelle Spijker)
4:  * This software is proprietary and confidential
5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6:  */
7:
8: #pragma once
9:
10: #include <exception>
11: #include <string>
12:
13: using namespace std;
14:
15: namespace Hardware {
16: namespace Exception {
17: class FailedToCreateThreadException : public std::exception {
18: public:
19:   FailedToCreateThreadException(string m = "Couldn't create the thread!")
20:       : msg(m){};
21:   ~FailedToCreateThreadException() _GLIBCXX_USE_NOEXCEPT{};
22:   const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };
23:
24: private:
25:   string msg;
26: };
27: }
28: }
```

```cpp
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "BBB.h"

namespace Hardware {
/*! Constructor*/
BBB::BBB() {
  threadRunning = false;
  callbackFunction = NULL;
  debounceTime = 0;
  thread = (pthread_t)NULL;
}

/*! De-constructor*/
BBB::~BBB() {}

/*! Reads the first line from a file
\param path constant string pointing towards the file
\returns this first line
*/
string BBB::Read(const string &path) {
  ifstream fs;
  fs.open(path.c_str());
  if (!fs.is_open()) {
    throw Exception::GPIOReadException(("Can't open: " + path).c_str());
  }
  string input;
  getline(fs, input);
  fs.close();
  return input;
}

/*! Writes a value to a file
\param path a constant string pointing towards the file
\param value a constant string which should be written in the file
*/
void BBB::Write(const string &path, const string &value) {
  ofstream fs;
  fs.open(path.c_str());
  if (!fs.is_open()) {
    throw Exception::GPIOReadException(("Can't open: " + path).c_str());
  }
  fs << value;
  fs.close();
}

/*! Checks if a directory exist
\returns true if the directory exists and false if not
*/
bool BBB::DirectoryExist(const string &path) {
  struct stat st;
  if (stat((char *)path.c_str(), &st) != 0) {
    return false;
  }
  return true;
}

/*! Checks if a cape is loaded in the file /sys/devices/bone_capemgr.9/slots
\param shield a const search string which is a (part) of the shield name
\return true if the search string is found otherwise false
*/
bool BBB::CapeLoaded(const string &shield) {
  bool shieldFound = false;

  ifstream fs;
  fs.open(SLOTS);
  if (!fs.is_open()) {
    throw Exception::GPIOReadException("Can't open SLOTS");
  }

  string line;
  while (getline(fs, line)) {
    if (line.find(shield) != string::npos) {
      shieldFound = true;
      break;
    }
  }
  fs.close();
```

```
   84:   return shieldFound;
   85: }
   86: }
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 * This code is based upon:
 * Derek Molloy, "Exploring BeagleBone: Tools and Techniques for Building
 * with Embedded Linux", Wiley, 2014, ISBN:9781118935125.
 * See: www.exploringbeaglebone.com
 */

#pragma once
#include "BBB.h"

#define EXPORT_PIN "/sys/class/gpio/export"
#define UNEXPORT_PIN "/sys/class/gpio/unexport"
#define GPIOS "/sys/class/gpio/gpio"
#define DIRECTION "/direction"
#define VALUE "/value"
#define EDGE "/edge"

using namespace std;

namespace Hardware {
class GPIO : public BBB {
public:
  enum Direction { Input, Output };
  enum Value { Low = 0, High = 1 };
  enum Edge { None, Rising, Falling, Both };

  int number; // Number of the pin

  int WaitForEdge();
  int WaitForEdge(CallbackType callback);
  void WaitForEdgeCancel() { this->threadRunning = false; }

  Value GetValue();
  void SetValue(Value value);

  Direction GetDirection();
  void SetDirection(Direction direction);

  Edge GetEdge();
  void SetEdge(Edge edge);

  GPIO(int number);
  ~GPIO();

private:
  string gpiopath;
  Direction direction;
  Edge edge;
  friend void *threadedPollGPIO(void *value);

  bool isExported(int number, Direction &dir, Edge &edge);
  bool ExportPin(int number);
  bool UnexportPin(int number);

  Direction ReadsDirection(const string &gpiopath);
  void WritesDirection(const string &gpiopath, Direction direction);

  Edge ReadsEdge(const string &gpiopath);
  void WritesEdge(const string &gpiopath, Edge edge);

  Value ReadsValue(const string &gpiopath);
  void WritesValue(const string &gpiopath, Value value);
};

void *threadedPollGPIO(void *value);
}
```

```
/* Copyright (C) Jelle Spijker - All Rights Reserved
 * Unauthorized copying of this file, via any medium is strictly prohibited
 * and only allowed with the written consent of the author (Jelle Spijker)
 * This software is proprietary and confidential
 * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
 */

#include "PWM.h"

namespace Hardware {
/// <summary>
/// Constructeur
/// </summary>
/// <param name="pin">Pin</param>
PWM::PWM(Pin pin) {
  this->pin = pin;

  // Check if PWM cape is loaded, if not load it
  if (!CapeLoaded(PWM_CAPE)) {
    Write(SLOTS, PWM_CAPE);
  }

  // Init the pin
  basepath = OCP_PATH;
  switch (pin) {
  case Hardware::PWM::P8_13:
    if (!CapeLoaded(P8_13_CAPE)) {
      Write(SLOTS, P8_13_CAPE_LOAD);
    }
    basepath.append(FindPath(P8_13_FIND));
    break;
  case Hardware::PWM::P8_19:
    if (!CapeLoaded(P8_19_CAPE)) {
      Write(SLOTS, P8_19_CAPE_LOAD);
    }
    basepath.append(FindPath(P8_19_FIND));
    break;
  case Hardware::PWM::P9_14:
    if (!CapeLoaded(P9_14_CAPE)) {
      Write(SLOTS, P9_14_CAPE_LOAD);
    }
    basepath.append(FindPath(P9_14_FIND));
    break;
  case Hardware::PWM::P9_16:
    if (!CapeLoaded(P9_16_CAPE)) {
      Write(SLOTS, P9_16_CAPE_LOAD);
    }
    basepath.append(FindPath(P9_16_FIND));
    break;
  }

  // Get the working paths
  dutypath = basepath + "/duty";
  periodpath = basepath + "/period";
  runpath = basepath + "/run";
  polaritypath = basepath + "/polarity";

  // Give Linux time to setup directory structure;
  usleep(250000);

  // Read current values
  period = StringToNumber<int>(Read(periodpath));
  duty = StringToNumber<int>(Read(dutypath));
  run = static_cast<Run>(StringToNumber<int>(Read(runpath)));
  polarity = static_cast<Polarity>(StringToNumber<int>(Read(polaritypath)));

  // calculate the current intensity
  calcIntensity();
}

PWM::~PWM() {}

/// <summary>
/// Calculate the current intensity
/// </summary>
void PWM::calcIntensity() {
  if (polarity == Normal) {
    if (duty == 0) {
      intensity = 0.0f;
    } else {
      intensity = (float)period / (float)duty;
    }
  } else {
```

```
   84:     if (period == 0) {
   85:       intensity = 0.0f;
   86:     } else {
   87:       intensity = (float)duty / (float)period;
   88:     }
   89:   }
   90: }
   91:
   92: /// <summary>
   93: /// Set the intensity level as percentage
   94: /// </summary>
   95: /// <param name="value">floating value multipication factor</param>
   96: void PWM::SetIntensity(float value) {
   97:   if (polarity == Normal) {
   98:     SetDuty(static_cast<int>((value * duty) + 0.5));
   99:   } else {
  100:     SetPeriod(static_cast<int>((value * period) + 0.5));
  101:   }
  102: }
  103:
  104: /// <summary>
  105: /// Set the output as a corresponding uint8_t value
  106: /// </summary>
  107: /// <param name="value">pixel value 0-255</param>
  108: void PWM::SetPixelValue(uint8_t value) {
  109:   if (period != 255) {
  110:     SetPeriod(255);
  111:   }
  112:   SetDuty(255 - value);
  113:   pixelvalue = value;
  114: }
  115:
  116: /// <summary>
  117: /// Set the period of the signal
  118: /// </summary>
  119: /// <param name="value">period : int</param>
  120: void PWM::SetPeriod(int value) {
  121:   string valstr = NumberToString<int>(value);
  122:   Write(periodpath, valstr);
  123:   period = value;
  124:
  125:   calcIntensity();
  126: }
  127:
  128: /// <summary>
  129: /// Set the duty of the signal
  130: /// </summary>
  131: /// <param name="value">duty : int</param>
  132: void PWM::SetDuty(int value) {
  133:   string valstr = NumberToString<int>(value);
  134:   Write(dutypath, valstr);
  135:   duty = value;
  136:
  137:   calcIntensity();
  138: }
  139:
  140: /// <summary>
  141: /// Run the signal
  142: /// </summary>
  143: /// <param name="value">On or Off</param>
  144: void PWM::SetRun(Run value) {
  145:   int valInt = static_cast<int>(value);
  146:   string valstr = NumberToString<int>(valInt);
  147:   Write(runpath, valstr);
  148:   run = value;
  149: }
  150:
  151: /// <summary>
  152: /// Set the polarity
  153: /// </summary>
  154: /// <param name="value">Normal or Inverted signal</param>
  155: void PWM::SetPolarity(Polarity value) {
  156:   int valInt = static_cast<int>(value);
  157:   string valstr = NumberToString<int>(valInt);
  158:   Write(runpath, valstr);
  159:   polarity = value;
  160: }
  161:
  162: /// <summary>
  163: /// Find the current PWM path in the OCP.3 directory
  164: /// </summary>
  165: /// <param name="value">part a the path name</param>
  166: /// <returns>Returns the first found value</returns>
```

```
167: string PWM::FindPath(string value) {
168:   auto dir = opendir(OCP_PATH);
169:   auto entity = readdir(dir);
170:   while (entity != NULL) {
171:     if (entity->d_type == DT_DIR) {
172:       string str = static_cast<string>(entity->d_name);
173:       if (str.find(value) != string::npos) {
174:         return str;
175:       }
176:     }
177:     entity = readdir(dir);
178:   }
179:   return "";
180: }
181: }
```