

```
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 "FailedToCreateGIOPollingThreadException.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 callbackfunction*/
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:  2:  * Unauthorized copying of this file, via any medium is strictly prohibited
3:  3:  * and only allowed with the written consent of the author (Jelle Spijker)
4:  4:  * This software is proprietary and confidential
5:  5:  * Written by Jelle Spijker <spijker.jelle@gmail.com>, 2015
6:  6:  */
7:
8:  8:  /*! \class EC12P
9:  9:  Interaction with the sparksfun RGB encoder
10: 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:     28:     /*! Enumerator indicating the color of the encoder shaft*/
29:     enum Color {
30:         Red,          30:         Red,          /*!< Red*/
31:         Pink,         31:         Pink,         /*!< Pink*/
32:         Blue,         32:         Blue,         /*!< Blue*/
33:         SkyBlue,      33:         SkyBlue,      /*!< SkyBlue*/
34:         Green,        34:         Green,        /*!< Green*/
35:         Yellow,       35:         Yellow,       /*!< Yellow*/
36:         White,        36:         White,        /*!< White*/
37:         None          37:         None          /*!< Off*/
38:     };
39:
40:     void SetPixelColor(Color value);
41:     Color GetPixelColor() { return PixelColor; };
42:
43:     void RainbowLoop(int sleeperperiod);
44:     void StopRainbowLoop() { threadRunning = false; };
45:
46:     eQEP Rotary{eQEP2, eQEP::eQEP_Mode_Absolute}; 46:     Rotary{eQEP2, eQEP::eQEP_Mode_Absolute}; /*!< The encoder*/
47:     GPIO Button{68}; 47:     Button{68};          /*!< The pushbutton*/
48:
49: private:
50:     Color PixelColor; 50:     PixelColor; /*!< Current shaft color*/
51:
52:     GPIO R{31}; 52:     R{31};          /*!< Red LED*/
53:     GPIO B{48}; 53:     B{48};          /*!< Blue LED*/
54:     GPIO G{51}; 54:     G{51};          /*!< Green LED*/
55:
56:     pthread_t thread; 56:     thread;          /*!< the thread*/
57:     bool threadRunning; 57:     threadRunning;   /*!< Bool used to stop the thread*/
58:     int sleeperperiod; 58:     sleeperperiod;   /*!< 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 sleeperperiod) {
83:      this->sleeperperiod = sleeperperiod;
```

```
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 through 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->sleeppperiod);
99:         i++;
100:         if (i == 6) {
101:             i = 0;
102:         }
103:     }
104:     return ec12p;
105: }
106: }
```

```

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 FailedToCreateGPiOPollingThreadException : public std::exception {
18:         public:
19:             FailedToCreateGPiOPollingThreadException(
20:                 string m = "Failed to create GPIO polling thread!")
21:                 : msg(m){};
22:             ~FailedToCreateGPiOPollingThreadException() _GLIBCXX_USE_NOEXCEPT{};
23:             const char *what() const _GLIBCXX_USE_NOEXCEPT { return msg.c_str(); };
24:
25:         private:
26:             string msg;
27:     };
28: }
29:

```



```
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 "USB.h"
9:
10: namespace Hardware {
11:     USB::USB() {}
12:
13:     USB::~USB() {}
14:
15:     void USB::ResetUSB() {
16:         int fd, rc;
17:
18:         fd = open("/dev/bus/usb/001/002", O_WRONLY);
19:         rc = ioctl(fd, USBDEVFS_RESET, 0);
20:         if (rc < 0) {
21:             throw - 1;
22:         }
23:         close(fd);
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:  #include "GPIO.h"
9:
10: namespace Hardware {
11:     GPIO::GPIO(int number) {
12:
13:         this->number = number;
14:         gpiopath = GPIOs + NumberToString<int>(number);
15:
16:         if (!isExported(number, direction, edge)) {
17:             ExportPin(number);
18:             direction = ReadsDirection(gpiopath);
19:             edge = ReadsEdge(gpiopath);
20:         }
21:         usleep(250000);
22:     }
23:
24:     GPIO::~GPIO() { UnexportPin(number); }
25:
26:     int GPIO::WaitForEdge(CallbackType callback) {
27:         threadRunning = true;
28:         callbackFunction = callback;
29:         if (pthread_create(&this->thread, NULL, &threadedPollGPIO,
30:             static_cast<void*>(this))) {
31:             threadRunning = false;
32:             throw Exception::FailedToCreateGIOPollingThreadException();
33:         }
34:         return 0;
35:     }
36:
37:     int GPIO::WaitForEdge() {
38:         if (direction == Output) {
39:             SetDirection(Input);
40:         }
41:         int fd, i, epollfd, count = 0;
42:         struct epoll_event ev;
43:         epollfd = epoll_create(1);
44:         if (epollfd == -1) {
45:             throw Exception::FailedToCreateGIOPollingThreadException(
46:                 "GPIO: Failed to create epollfd!");
47:         }
48:         if ((fd = open((gpiopath + VALUE).c_str(), O_RDONLY | O_NONBLOCK)) == -1) {
49:             throw Exception::GPIOReadException();
50:         }
51:
52:         // read operation | edge triggered | urgent data
53:         ev.events = EPOLLIN | EPOLLET | EPOLLPRI;
54:         ev.data.fd = fd;
55:
56:         if (epoll_ctl(epollfd, EPOLL_CTL_ADD, fd, &ev) == -1) {
57:             throw Exception::FailedToCreateGIOPollingThreadException(
58:                 "GPIO: Failed to add control interface!");
59:         }
60:
61:         while (count <= 1) {
62:             i = epoll_wait(epollfd, &ev, 1, -1);
63:             if (i == -1) {
64:                 close(fd);
65:                 return -1;
66:             } else {
67:                 count++;
68:             }
69:         }
70:         close(fd);
71:         return 0;
72:     }
73:
74:     GPIO::Value GPIO::GetValue() { return ReadsValue(gpiopath); }
75:     void GPIO::SetValue(GPIO::Value value) { WritesValue(gpiopath, value); }
76:
77:     GPIO::Direction GPIO::GetDirection() { return direction; }
78:     void GPIO::SetDirection(Direction direction) {
79:         this->direction = direction;
80:         WritesDirection(gpiopath, direction);
81:     }
82:
83:     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: }
```

```
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 MicroscopeNotFoundException : public std::exception {
17: public:
18:     MicroscopeNotFoundException(string m = "Microscope not found exception!")
19:         : msg(m){};
20:     ~MicroscopeNotFoundException() _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:  #include "Microscope.h"
9:
10: // Custom exceptions
11: #include "MicroscopeNotFoundException.h"
12: #include "CouldNotGrabImageException.h"
13:
14: using namespace cv;
15: using namespace boost::filesystem;
16: using namespace std;
17:
18: namespace Hardware {
19:
20:  /*! Constructor of the class which initializes the USB microscope
21:  \param frameDelayTrigger the delay between the first initialization of the
22:  microscope and the retrieval of the image expressed in seconds. Default value is
23:  3 seconds
24:  \param dimension A resolution Struct indicating which resolution the webcam
25:  should use. Default is 2592 x 1944
26:  */
27:  Microscope::Microscope(uint8_t frameDelayTrigger, Resolution dimensions,
28:                          bool firstdefault) {
29:      FrameDelayTrigger = frameDelayTrigger;
30:      Dimensions = dimensions;
31:
32:      StartupSeq(firstdefault);
33:  }
34:
35:  void Microscope::StartupSeq(bool firstdefault) {
36:      std::vector<std::string> camNames = AvailableCams();
37:      uint videodev = find(camNames.begin(), camNames.end(), MICROSCOPE_NAME) -
38:                      camNames.begin();
39:      if (videodev == camNames.size() && !firstdefault) {
40:          throw Exception::MicroscopeNotFoundException(
41:              "Microscope Not Found Exception!");
42:      } else if (videodev == camNames.size() && firstdefault) {
43:          videodev = 0;
44:      }
45:
46:      struct utsname unameData;
47:      uname(&unameData);
48:      arch = static_cast<std::string>(unameData.machine);
49:
50:      try {
51:          openCam(videodev);
52:      } catch (Exception::MicroscopeNotFoundException &e) {
53:          // Tries to soft reset the USB port. Haven't got this working yet
54:          USB usbdev;
55:          usbdev.ResetUSB();
56:          captureDevice.open(videodev);
57:          if (!captureDevice.isOpened()) {
58:              throw Exception::MicroscopeNotFoundException("USB Soft Reset Exception!");
59:          }
60:      }
61:  }
62:
63:  /*!< De-constructor*/
64:  Microscope::~Microscope() { captureDevice.~VideoCapture(); }
65:
66:  /*! Get the frame after the set initialization period
67:  \param dst a cv::Mat construct which stores the retrieved image
68:  */
69:  void Microscope::GetFrame(cv::Mat &dst) {
70:      // Work around for crappy cam retrieval of the BBB
71:      if (arch.find("armv7l") != string::npos) {
72:          if (!captureDevice.grab()) {
73:              throw Exception::CouldNotGrabImageException();
74:          }
75:          sleep(FrameDelayTrigger); // Needed otherwise scrambled picture
76:          if (!captureDevice.grab()) {
77:              throw Exception::CouldNotGrabImageException();
78:          }
79:          captureDevice.retrieve(dst);
80:      } else {
81:          if (!captureDevice.read(dst)) {
82:              throw Exception::CouldNotGrabImageException();
83:          }
84:      }
85:  }

```



```

84:     }
85: }
86:
87: /*! Get an HDR capture of the cam using a user defined number of frames
88: 88: differently lit frames. Due to hardware limitations each frames take roughly 3
89: 89: seconds to grab. This function is based upon the tutorial from openCV
90: 90: http://docs.opencv.org/trunk/doc/tutorials/photo/hdr_imaging/hdr_imaging.html
91: 91: \param dst a cv::Mat construct with the retrieved HDR result
92: 92: \param noframes is the number of frames that create the HDR image - default = 5
93: 93: */
94: void Microscope::GetHDRFrame(cv::Mat &dst, uint32_t noframes) {
95:     prog_sig(0);
96:     // create the brightness steps
97:     int8_t brightnessStep =
98:         static_cast<int8_t>((MAX_BRIGHTNESS - MIN_BRIGHTNESS) / noframes);
99:     int8_t currentBrightness = captureDevice.get(CV_CAP_PROP_BRIGHTNESS);
100:    int8_t currentContrast = captureDevice.get(CV_CAP_PROP_CONTRAST);
101:    captureDevice.set(CV_CAP_PROP_CONTRAST, MAX_CONTRAST);
102:
103:    int progStep = 70 / noframes;
104:    Mat currentImg;
105:    // take the shots at different brightness levels
106:    for (uint32_t i = 1; i <= noframes; i++) {
107:        captureDevice.set(CV_CAP_PROP_BRIGHTNESS,
108:            (MIN_BRIGHTNESS + (i * brightnessStep)));
109:        GetFrame(currentImg);
110:        HDRframes.push_back(currentImg);
111:        prog_sig(i * progStep);
112:    }
113:
114:    // Set the brightness and back to the previous used level
115:    captureDevice.set(CV_CAP_PROP_BRIGHTNESS, currentBrightness);
116:    captureDevice.set(CV_CAP_PROP_CONTRAST, currentContrast);
117:
118:    // Perform the exposure fusion
119:    Mat fusion;
120:    Ptr<MergeMertens> merge_mertens = createMergeMertens();
121:    merge_mertens->process(HDRframes, fusion);
122:    prog_sig(80);
123:    fusion *= 255;
124:    prog_sig(85);
125:    fusion.convertTo(dst, CV_8UC1);
126:    prog_sig(100);
127:    fin_sig();
128: }
129:
130: /*!< Checks if the capture device is open and returns the status as a bool
131: 131: /return Status of the capture device expressed as a bool
132: 132: */
133: bool Microscope::IsOpened() { return captureDevice.isOpened(); }
134:
135: /*!< Safely release the capture device*/
136: void Microscope::Release() { captureDevice.release(); }
137:
138: /*!< Opens the webcam*/
139: void Microscope::openCam(int dev) {
140:     captureDevice.open(dev);
141:     if (!captureDevice.isOpened()) {
142:         throw Exception::MicroscopeNotFoundException();
143:     }
144:     captureDevice.set(CV_CAP_PROP_FRAME_WIDTH, Dimensions.Width);
145:     captureDevice.set(CV_CAP_PROP_FRAME_HEIGHT, Dimensions.Height);
146: }
147:
148: std::vector<std::string> Microscope::AvailableCams() {
149:     std::vector<std::string> cams;
150:     const string path_ss = "/sys/class/video4linux";
151:
152:     if (!exist(path_ss))
153:         return cams;
154:
155:     for (directory_iterator itr(path_ss); itr != directory_iterator(); ++itr) {
156:         string videoln = itr->path().string();
157:         videoln.append("/name");
158:         if (exist(videoln)) {
159:             std::ifstream camName;
160:             camName.open(videoln);
161:             std::string name;
162:             std::getline(camName, name);
163:             cams.push_back(name);
164:             camName.close();
165:         }
166:     }

```

```
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: }
```

```
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 "BBB.h"
10: #include <dirent.h>
11:
12: #define OCP_PATH "/sys/devices/ocp.3/"
13: #define P8_13_FIND "bs_pwm_test_P8_13"
14: #define P8_19_FIND "bs_pwm_test_P8_19"
15: #define P9_14_FIND "bs_pwm_test_P9_14"
16: #define P9_16_FIND "bs_pwm_test_P9_16"
17:
18: #define PWM_CAPE "am33xx_pwm"
19: #define P8_13_CAPE "bspwm_P8_13" //_14
20: #define P8_19_CAPE "bspwm_P8_13" //_14
21: #define P9_14_CAPE "bspwm_P9_14" //_16
22: #define P9_16_CAPE "bspwm_P9_16" //_16
23:
24: #define P8_13_CAPE_LOAD "bspwm_P8_13_14"
25: #define P8_19_CAPE_LOAD "bspwm_P8_13_14"
26: #define P9_14_CAPE_LOAD "bspwm_P9_14_16"
27: #define P9_16_CAPE_LOAD "bspwm_P9_16_16"
28:
29: namespace Hardware {
30: class PWM : public BBB {
31: public:
32:     enum Pin // Four possible PWM pins
33:     { P8_13,
34:       P8_19,
35:       P9_14,
36:       P9_16 };
37:     enum Run // Signal generating
38:     { On = 1,
39:       Off = 0 };
40:     enum Polarity // Inverse duty polarity
41:     { Normal = 1,
42:       Inverted = 0 };
43:
44:     Pin pin; // Current pin
45:
46:     uint8_t GetPixelValue() { return pixelvalue; }
47:     void SetPixelValue(uint8_t value);
48:
49:     float GetIntensity() { return intensity; };
50:     void SetIntensity(float value);
51:
52:     int GetPeriod() { return period; };
53:     void SetPeriod(int value);
54:
55:     int GetDuty() { return duty; };
56:     void SetDuty(int value);
57:     void SetIntensity();
58:
59:     Run GetRun() { return run; };
60:     void SetRun(Run value);
61:
62:     Polarity GetPolarity() { return polarity; };
63:     void SetPolarity(Polarity value);
64:
65:     PWM(Pin pin);
66:     ~PWM();
67:
68: private:
69:     int period; // current period
70:     int duty; // current duty
71:     float intensity; // current intensity
72:     uint8_t pixelvalue; // current pixelvalue
73:     Run run; // current run state
74:     Polarity polarity; // current polaity
75:
76:     string basepath; // the basepath ocp.3
77:     string dutypath; // base + duty path
78:     string periodpath; // base + period path
79:     string runpath; // base + run path
80:     string polaritypath; // base + polarity path
81:
82:     void calcIntensity();
83:     string FindPath(string value);
```



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

```
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 "SoilCape.h"
9:
10: namespace Hardware {
11:   SoilCape::SoilCape() {}
12:
13:   SoilCape::~SoilCape() {}
14: }
```

```
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: }
```

```
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 "ADC.h"
9:
10: namespace Hardware {
11:  /*! Constructor
12:  \param pin and ADCPin type indicating which analogue pin to use
13:  */
14:  ADC::ADC(ADCPin pin) {
15:      this->Pin = pin;
16:      switch (pin) {
17:          case Hardware::ADC::ADC0:
18:              adcpath = ADC0_PATH;
19:              break;
20:          case Hardware::ADC::ADC1:
21:              adcpath = ADC1_PATH;
22:              break;
23:          case Hardware::ADC::ADC2:
24:              adcpath = ADC2_PATH;
25:              break;
26:          case Hardware::ADC::ADC3:
27:              adcpath = ADC3_PATH;
28:              break;
29:          case Hardware::ADC::ADC4:
30:              adcpath = ADC4_PATH;
31:              break;
32:          case Hardware::ADC::ADC5:
33:              adcpath = ADC5_PATH;
34:              break;
35:          case Hardware::ADC::ADC6:
36:              adcpath = ADC6_PATH;
37:              break;
38:          case Hardware::ADC::ADC7:
39:              adcpath = ADC7_PATH;
40:              break;
41:      }
42:
43:      MinIntensity = 0;
44:      MaxIntensity = 4096;
45:  }
46:
47:  /*! De-constructor*/
48:  ADC::~ADC() {}
49:
50:  /*! Reads the current voltage in the pin
51:  \return an integer between 0 and 4096
52:  */
53:  int ADC::GetCurrentValue() {
54:      int retVal = StringToNumber<int>(Read(adcpath));
55:      Intensity = (float)(retVal - MinIntensity) /
56:                  (4096 - (MinIntensity + (4096 - MaxIntensity)));
57:      return retVal;
58:  }
59:
60:  /*! Set the current voltage at the pin as the minimum voltage*/
61:  void ADC::SetMinIntensity() {
62:      MinIntensity = StringToNumber<int>(Read(adcpath));
63:  }
64:
65:  void ADC::SetMaxIntensity() {
66:      MaxIntensity = StringToNumber<int>(Read(adcpath));
67:  }
68:
69:  /*! Threading enabled polling of the analogue pin
70:  \param callback the function which should be called when polling indicates a
71:  change CallbackType
72:  \return 0
73:  */
74:  int ADC::WaitForValueChange(CallbackType callback) {
75:      threadRunning = true;
76:      callbackFunction = callback;
77:      if (pthread_create(&thread, NULL, &threadedPollADC,
78:                      static_cast<void*>(this))) {
79:          threadRunning = false;
80:          throw Exception::FailedToCreateGPIOPollingThreadException();
81:      }
82:      return 0;
83:  }
```

```
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: }
```

```
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 ADC
9:  Interaction with the beaglebone analogue pins
10: */
11:
12: #pragma once
13:
14: #include "BBB.h"
15: #include "ADCReadException.h"
16:
17: #define ADC0_PATH \
18:     "/sys/bus/iio/devices/iio:device0/in_voltage0_raw" /*!< path to analogue pin \
19:                                                         0*/
20: #define ADC1_PATH \
21:     "/sys/bus/iio/devices/iio:device0/in_voltage1_raw" /*!< path to analogue pin \
22:                                                         1*/
23: #define ADC2_PATH \
24:     "/sys/bus/iio/devices/iio:device0/in_voltage2_raw" /*!< path to analogue pin \
25:                                                         2*/
26: #define ADC3_PATH \
27:     "/sys/bus/iio/devices/iio:device0/in_voltage3_raw" /*!< path to analogue pin \
28:                                                         3*/
29: #define ADC4_PATH \
30:     "/sys/bus/iio/devices/iio:device0/in_voltage4_raw" /*!< path to analogue pin \
31:                                                         4*/
32: #define ADC5_PATH \
33:     "/sys/bus/iio/devices/iio:device0/in_voltage5_raw" /*!< path to analogue pin \
34:                                                         5*/
35: #define ADC6_PATH \
36:     "/sys/bus/iio/devices/iio:device0/in_voltage6_raw" /*!< path to analogue pin \
37:                                                         6*/
38: #define ADC7_PATH \
39:     "/sys/bus/iio/devices/iio:device0/in_voltage7_raw" /*!< path to analogue pin \
40:                                                         7*/
41:
42: namespace Hardware {
43: class ADC : public BBB {
44: public:
45:     /*! Enumerator to indicate the analogue pin*/
46:     enum ADCPin {
47:         ADC0, /*!< AIN0 pin*/
48:         ADC1, /*!< AIN1 pin*/
49:         ADC2, /*!< AIN2 pin*/
50:         ADC3, /*!< AIN3 pin*/
51:         ADC4, /*!< AIN4 pin*/
52:         ADC5, /*!< AIN5 pin*/
53:         ADC6, /*!< AIN6 pin*/
54:         ADC7 /*!< AIN7 pin*/
55:     };
56:
57:     ADCPin Pin; /*!< current pin*/
58:
59:     ADC(ADCPin pin);
60:     ~ADC();
61:
62:     int GetCurrentValue();
63:     float GetIntensity() { return Intensity; }
64:     int GetMinIntensity() { return MinIntensity; }
65:     int GetMaxIntensity() { return MaxIntensity; }
66:
67:     void SetMinIntensity();
68:     void SetMaxIntensity();
69:
70:     int WaitForValueChange();
71:     int WaitForValueChange(CallbackType callback);
72:     void WaitForValueChangeCancel() { this->threadRunning = false; }
73:
74: private:
75:     string adcpath; /*!< Path to analogue write file*/
76:     float Intensity; /*!< Current intensity expressed as percentage*/
77:     int MinIntensity; /*!< Voltage level which represent 0 percentage*/
78:     int MaxIntensity; /*!< Voltage level which represent 100 percentage*/
79:
80:     friend void *threadedPollADC(void *value); /*!< friend polling function*/
81: };
82:
83: 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 capabilities
22: *
23: */
24:
25: // Pull in our eQEP driver definitions
26: #include "eqep.h"
27:
28: // Language dependencies
29: #include <stdint>
30: #include <stdlib>
31: #include <stdio>
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: }

```

```

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 "BBB.h"
9:
10: namespace Hardware {
11:  /* Constructor*/
12:  BBB::BBB() {
13:      threadRunning = false;
14:      callbackFunction = NULL;
15:      debounceTime = 0;
16:      thread = (pthread_t)NULL;
17:  }
18:
19:  /* De-constructor*/
20:  BBB::~BBB() {}
21:
22:  /* Reads the first line from a file
23:  \param path constant string pointing towards the file
24:  \returns this first line
25:  */
26:  string BBB::Read(const string &path) {
27:      ifstream fs;
28:      fs.open(path.c_str());
29:      if (!fs.is_open()) {
30:          throw Exception::GPIOReadException(("Can't open: " + path).c_str());
31:      }
32:      string input;
33:      getline(fs, input);
34:      fs.close();
35:      return input;
36:  }
37:
38:  /* Writes a value to a file
39:  \param path a constant string pointing towards the file
40:  \param value a constant string which should be written in the file
41:  */
42:  void BBB::Write(const string &path, const string &value) {
43:      ofstream fs;
44:      fs.open(path.c_str());
45:      if (!fs.is_open()) {
46:          throw Exception::GPIOReadException(("Can't open: " + path).c_str());
47:      }
48:      fs << value;
49:      fs.close();
50:  }
51:
52:  /* Checks if a directory exist
53:  \returns true if the directory exists and false if not
54:  */
55:  bool BBB::DirectoryExist(const string &path) {
56:      struct stat st;
57:      if (stat((char *)path.c_str(), &st) != 0) {
58:          return false;
59:      }
60:      return true;
61:  }
62:
63:  /* Checks if a cape is loaded in the file /sys/devices/bone_capemgr.9/slots
64:  \param shield a const search string which is a (part) of the shield name
65:  \return true if the search string is found otherwise false
66:  */
67:  bool BBB::CapeLoaded(const string &shield) {
68:      bool shieldFound = false;
69:
70:      ifstream fs;
71:      fs.open(SLOTS);
72:      if (!fs.is_open()) {
73:          throw Exception::GPIOReadException("Can't open SLOTS");
74:      }
75:
76:      string line;
77:      while (getline(fs, line)) {
78:          if (line.find(shield) != string::npos) {
79:              shieldFound = true;
80:              break;
81:          }
82:      }
83:      fs.close();

```

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

```
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:  * This code is based upon:
7:  * Derek Molloy, "Exploring BeagleBone: Tools and Techniques for Building
8:  * with Embedded Linux", Wiley, 2014, ISBN:9781118935125.
9:  * See: www.exploringbeaglebone.com
10: */
11:
12: #pragma once
13: #include "BBB.h"
14:
15: #define EXPORT_PIN "/sys/class/gpio/export"
16: #define UNEXPORT_PIN "/sys/class/gpio/unexport"
17: #define GPIOS "/sys/class/gpio/gpio"
18: #define DIRECTION "/direction"
19: #define VALUE "/value"
20: #define EDGE "/edge"
21:
22: using namespace std;
23:
24: namespace Hardware {
25:     class GPIO : public BBB {
26:     public:
27:         enum Direction { Input, Output };
28:         enum Value { Low = 0, High = 1 };
29:         enum Edge { None, Rising, Falling, Both };
30:
31:         int number; // Number of the pin
32:
33:         int WaitForEdge();
34:         int WaitForEdge(CallbackType callback);
35:         void WaitForEdgeCancel() { this->threadRunning = false; }
36:
37:         Value GetValue();
38:         void SetValue(Value value);
39:
40:         Direction GetDirection();
41:         void SetDirection(Direction direction);
42:
43:         Edge GetEdge();
44:         void SetEdge(Edge edge);
45:
46:         GPIO(int number);
47:         ~GPIO();
48:
49:     private:
50:         string gpiopath;
51:         Direction direction;
52:         Edge edge;
53:         friend void *threadedPollGPIO(void *value);
54:
55:         bool isExported(int number, Direction &dir, Edge &edge);
56:         bool ExportPin(int number);
57:         bool UnexportPin(int number);
58:
59:         Direction ReadsDirection(const string &gpiopath);
60:         void WritesDirection(const string &gpiopath, Direction direction);
61:
62:         Edge ReadsEdge(const string &gpiopath);
63:         void WritesEdge(const string &gpiopath, Edge edge);
64:
65:         Value ReadsValue(const string &gpiopath);
66:         void WritesValue(const string &gpiopath, Value value);
67:     };
68:
69: void *threadedPollGPIO(void *value);
70: }
```

```
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 "PWM.h"
9:
10: namespace Hardware {
11:  /// <summary>
12:  /// Constructeur
13:  /// </summary>
14:  /// <param name="pin">Pin</param>
15:  PWM::PWM(Pin pin) {
16:      this->pin = pin;
17:
18:      // Check if PWM cape is loaded, if not load it
19:      if (!CapeLoaded(PWM_CAPE)) {
20:          Write(SLOTS, PWM_CAPE);
21:      }
22:
23:      // Init the pin
24:      basepath = OCP_PATH;
25:      switch (pin) {
26:          case Hardware::PWM::P8_13:
27:              if (!CapeLoaded(P8_13_CAPE)) {
28:                  Write(SLOTS, P8_13_CAPE_LOAD);
29:              }
30:              basepath.append(FindPath(P8_13_FIND));
31:              break;
32:          case Hardware::PWM::P8_19:
33:              if (!CapeLoaded(P8_19_CAPE)) {
34:                  Write(SLOTS, P8_19_CAPE_LOAD);
35:              }
36:              basepath.append(FindPath(P8_19_FIND));
37:              break;
38:          case Hardware::PWM::P9_14:
39:              if (!CapeLoaded(P9_14_CAPE)) {
40:                  Write(SLOTS, P9_14_CAPE_LOAD);
41:              }
42:              basepath.append(FindPath(P9_14_FIND));
43:              break;
44:          case Hardware::PWM::P9_16:
45:              if (!CapeLoaded(P9_16_CAPE)) {
46:                  Write(SLOTS, P9_16_CAPE_LOAD);
47:              }
48:              basepath.append(FindPath(P9_16_FIND));
49:              break;
50:      }
51:
52:      // Get the working paths
53:      dutypath = basepath + "/duty";
54:      periodpath = basepath + "/period";
55:      runpath = basepath + "/run";
56:      polaritypath = basepath + "/polarity";
57:
58:      // Give Linux time to setup directory structure;
59:      usleep(250000);
60:
61:      // Read current values
62:      period = StringToNumber<int>(Read(periodpath));
63:      duty = StringToNumber<int>(Read(dutypath));
64:      run = static_cast<Run>(StringToNumber<int>(Read(runpath)));
65:      polarity = static_cast<Polarity>(StringToNumber<int>(Read(polaritypath)));
66:
67:      // calculate the current intensity
68:      calcIntensity();
69:  }
70:
71:  PWM::~PWM() {}
72:
73:  /// <summary>
74:  /// Calculate the current intensity
75:  /// </summary>
76:  void PWM::calcIntensity() {
77:      if (polarity == Normal) {
78:          if (duty == 0) {
79:              intensity = 0.0f;
80:          } else {
81:              intensity = (float)period / (float)duty;
82:          }
83:      } 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 multiplication 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: }
```