```cpp
/*
* TI eQEP driver interface API
*
* Copyright (C) 2013 Nathaniel R. Lewis - http://nathanielrlewis.com/
*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
* GNU General Public License for more details.
*
* You should have received a copy of the GNU General Public License
* along with this program; if not, write to the Free Software
* Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
*
*/

// Pull in our eQEP driver definitions
#include "eqep.h"

// Language dependencies
#include <cstdint>
#include <cstdlib>
#include <cstdio>

// POSIX dependencies
#include <unistd.h>
#include <fcntl.h>
#include <poll.h>
#include <sys/types.h>
#include <sys/stat.h>

namespace Hardware
{
    // Constructor for eQEP driver interface object
    eQEP::eQEP(std::string _path, eQEP::eQEP_Mode _mode)
        : path(_path)
```

```
42      {
43          if (_path == eQEP0)
44          {
45              if (!CapeLoaded("bone_eqep0")) { Write(SLOTS, "bone_eqep0"); }
46          }
47          else if (_path == eQEP1)
48          {
49              if (!CapeLoaded("bone_eqep1")) { Write(SLOTS, "bone_eqep1"); }
50          }
51          else if (_path == eQEP2)
52          {
53              if (!CapeLoaded("bone_eqep2b")) { Write(SLOTS, "bone_eqep2b"); }
54          }
55
56          // Set the mode of the hardware
57          this->set_mode(_mode);
58
59          // Reset the position
60          this->set_position(0);
61      }
62
63      // Set the position of the eQEP hardware
64      void eQEP::set_position(int32_t position)
65      {
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          {
72              // Error, break out
73              std::cerr << "[eQEP " << this->path << "] Unable to open position for write" << 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 mode of the eQEP hardware
```

```cpp
void eQEP::set_period(uint64_t period)
{
    // Open the file representing the position
    FILE *fp = fopen((this->path + "/period").c_str(), "w");

    // Check that we opened the file correctly
    if (fp == NULL)
    {
        // Error, break out
        std::cerr << "[eQEP " << this->path << "] Unable to open period for write" << std::endl;
        return;
    }

    // Write the desired value to the file
    fprintf(fp, "%llu\n", period);

    // Commit changes
    fclose(fp);
}

// Set the mode of the eQEP hardware
void eQEP::set_mode(eQEP::eQEP_Mode _mode)
{
    // Open the file representing the position
    FILE *fp = fopen((this->path + "/mode").c_str(), "w");

    // Check that we opened the file correctly
    if (fp == NULL)
    {
        // Error, break out
        std::cerr << "[eQEP " << this->path << "] Unable to open mode for write" << std::endl;
        return;
    }

    // Write the desired value to the file
    fprintf(fp, "%u\n", _mode);

    // Commit changes
    fclose(fp);
}

int eQEP::WaitForPositionChange(CallbackType callback)
```

```
127     {
128         threadRunning = true;
129         callbackFunction = callback;
130         if (pthread_create(&this->thread, NULL, &threadedPolleqep, static_cast<void*>(this)))
131         {
132             threadRunning = false;
133             throw Exception::FailedToCreateGPIOPollingThreadException();
134         }
135 
136         return 0;
137     }
138 
139     // Get the position of the hardware
140     int32_t eQEP::get_position(bool _poll)
141     {
142         // Position temporary variable
143         int32_t       position;
144         char          dummy;
145         struct pollfd ufd;
146 
147         // Do we want to poll?
148         if (_poll)
149         {
150             // Open a connection to the attribute file.
151             if ((ufd.fd = open((this->path + "/position").c_str(), O_RDWR)) < 0)
152             {
153                 // Error, break out
154                 std::cerr << "[eQEP " << this->path << "] unable to open position for polling" << std::endl;
155                 return 0;
156             }
157 
158             // Dummy read
159             read(ufd.fd, &dummy, 1);
160 
161             // Poll the port
162             ufd.events = (short)EPOLLET;
163             if (poll(&ufd, 1, -1) < 0)
164             {
165                 // Error, break out
166                 std::cerr << "[eQEP " << this->path << "] Error occurred whilst polling" << std::endl;
167                 close(ufd.fd);
168                 return 0;
```

```
169             }
170         }
171
172         // Read the position
173         FILE *fp = fopen((this->path + "/position").c_str(), "r");
174
175         // Check that we opened the file correctly
176         if (fp == NULL)
177         {
178             // Error, break out
179             std::cerr << "[eQEP " << this->path << "] Unable to open position for read" << std::endl;
180             close(ufd.fd);
181             return 0;
182         }
183
184         // Write the desired value to the file
185         fscanf(fp, "%d", &position);
186
187         // Commit changes
188         fclose(fp);
189
190         // If we were polling, close the polling file
191         if (_poll)
192         {
193             close(ufd.fd);
194         }
195
196         // Return the position
197         return position;
198     }
199
200     // Get the period of the eQEP hardware
201     uint64_t eQEP::get_period()
202     {
203         // Open the file representing the position
204         FILE *fp = fopen((this->path + "/period").c_str(), "r");
205
206         // Check that we opened the file correctly
207         if (fp == NULL)
208         {
209             // Error, break out
210             std::cerr << "[eQEP " << this->path << "] Unable to open period for read" << std::endl;
```

```
211             return 0;
212         }
213
214         // Write the desired value to the file
215         uint64_t period = 0;
216         fscanf(fp, "%llu", &period);
217
218         // Commit changes
219         fclose(fp);
220
221         // Return the period
222         return period;
223     }
224
225     // Get the mode of the eQEP hardware
226     eQEP::eQEP_Mode eQEP::get_mode()
227     {
228         // Open the file representing the position
229         FILE *fp = fopen((this->path + "/mode").c_str(), "r");
230
231         // Check that we opened the file correctly
232         if (fp == NULL)
233         {
234             // Error, break out
235             std::cerr << "[eQEP " << this->path << "] Unable to open mode for read" << std::endl;
236             return eQEP::eQEP_Mode_Error;
237         }
238
239         // Write the desired value to the file
240         eQEP::eQEP_Mode mode;
241         fscanf(fp, "%u", (unsigned int*)&mode);
242
243         // Commit changes
244         fclose(fp);
245
246         // Return the mode
247         return mode;
248     }
249
250     void* threadedPolleqep(void *value)
251     {
252         eQEP *eqep = static_cast<eQEP*>(value);
```

```
253         while (eqep->threadRunning)
254         {
255             eqep->callbackFunction(eqep->get_position(true));
256             usleep(eqep->debounceTime * 1000);
257         }
258         return 0;
259     }
260
261 }
```