GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
File:	storage/blockdevice/source/BufferedBlockDevice.cpp	Lines:	155	173	89.6 %
Date:	2021-05-06 12:39:05	Branches:	92	132	69.7 %

Line	Branch	Exec	Source
1			/* mbed Microcontroller Library
2			* Copyright (c) 2018 ARM Limited
3			* SPDX-License-Identifier: Apache-2.0
4			*
5			* Licensed under the Apache License, Version 2.0 (the "License");
6			* you may not use this file except in compliance with the License.
7			* You may obtain a copy of the License at
8			*
9			* http://www.apache.org/licenses/LICENSE-2.0
10			*
11			* Unless required by applicable law or agreed to in writing, software
12			* distributed under the License is distributed on an "AS IS" BASIS,
13			* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14			* See the License for the specific language governing permissions and
15			* limitations under the License.
16			*/
17			
18			#include "blockdevice/BufferedBlockDevice.h"
19			#include "platform/mbed_assert.h"
20			#include "platform/mbed_atomic.h"
21			#include <algorithm>
22			#include <string.h>
23			
24			namespace mbed {
25			
26		153965	static inline uint32_t align_down(bd_size_t val, bd_size_t size)
27			{
28		153965	return val / size * size;
29			}
30			
31		10221	BufferedBlockDevice::BufferedBlockDevice(BlockDevice *bd)
32			: _bd(bd), _bd_program_size(0), _bd_read_size(0), _bd_size(0), _write_cache_addr(0), _write_cache_valid(false),

```
33                10221  {     _write_cache(0), _read_buf(0), _init_ref_count(0), _is_initialized(false)
34
35  ✗✓✗✗          10221      MBED_ASSERT(_bd);
36                10221  }
37
38                30651  BufferedBlockDevice::~BufferedBlockDevice()
39                       {
40                10221      deinit();
41                20430  }
42
43                10221  int BufferedBlockDevice::init()
44                       {
45                10221      uint32_t val = core_util_atomic_incr_u32(&_init_ref_count, 1);
46
47      ✗✓        10221      if (val != 1) {
48                               return BD_ERROR_OK;
49                           }
50
51                10221      int err = _bd->init();
52      ✗✓        10221      if (err) {
53                               return err;
54                           }
55
56                10221      _bd_read_size = _bd->get_read_size();
57                10221      _bd_program_size = _bd->get_program_size();
58                10221      _bd_size = _bd->size();
59
60      ✓✗        10221      if (!_write_cache) {
61                10221          _write_cache = new uint8_t[_bd_program_size];
62                           }
63
64      ✓✗        10221      if (!_read_buf) {
65                10221          _read_buf = new uint8_t[_bd_read_size];
66                           }
67
68                10221      invalidate_write_cache();
69
70                10221      _is_initialized = true;
71                10221      return BD_ERROR_OK;
72                       }
73
74                20442  int BufferedBlockDevice::deinit()
75                       {
```

```
76  ✓✓  20442      if (!_is_initialized) {
77      10221          return BD_ERROR_OK;
78                  }
79
80                  // Flush out all data from buffers
81      10221       int err = sync();
82  ✗✓  10221       if (err) {
83                      return err;
84                  }
85
86      10221       uint32_t val = core_util_atomic_decr_u32(&_init_ref_count, 1);
87
88  ✗✓  10221       if (val) {
89                      return BD_ERROR_OK;
90                  }
91
92  ✓✗  10221       delete[] _write_cache;
93      10221       _write_cache = 0;
94  ✓✗  10221       delete[] _read_buf;
95      10221       _read_buf = 0;
96      10221       _is_initialized = false;
97      10221       return _bd->deinit();
98              }
99
100     71895   int BufferedBlockDevice::flush()
101             {
102 ✗✓  71895       MBED_ASSERT(_write_cache);
103 ✗✓  71895       if (!_is_initialized) {
104                     return BD_ERROR_DEVICE_ERROR;
105                 }
106
107 ✓✓  71895       if (_write_cache_valid) {
108     21253           int ret = _bd->program(_write_cache, _write_cache_addr, _bd_program_size);
109 ✗✓  21253           if (ret) {
110                         return ret;
111                     }
112     21253           invalidate_write_cache();
113                 }
114     71895       return 0;
115             }
116
117     61270   void BufferedBlockDevice::invalidate_write_cache()
```

```
118                 {
119      61270          _write_cache_addr = _bd_size;
120      61270          _write_cache_valid = false;
121      61270      }
122
123      20847      int BufferedBlockDevice::sync()
124                 {
125   ✓✓ 20847          if (!_is_initialized) {
126          1              return BD_ERROR_DEVICE_ERROR;
127                     }
128
129   ✗✓ 20846          MBED_ASSERT(_write_cache);
130      20846          int ret = flush();
131   ✗✓ 20846          if (ret) {
132                          return ret;
133                     }
134      20846          return _bd->sync();
135                 }
136
137    1703493      int BufferedBlockDevice::read(void *b, bd_addr_t addr, bd_size_t size)
138                 {
139   ✓✓ 1703493        if (!_is_initialized) {
140          1              return BD_ERROR_DEVICE_ERROR;
141                     }
142
143 ✓✗✗✓ 1703492        MBED_ASSERT(_write_cache && _read_buf);
144
145   ✓✓ 1703492        if (!is_valid_read(addr, size)) {
146          1              return BD_ERROR_DEVICE_ERROR;
147                     }
148
149                     // Common case - no need to involve write cache or read buffer
150 ✓✓✓✓ 1704713        if (_bd->is_valid_read(addr, size) &&
     ✓✓
151   ✗✓     614                ((addr + size <= _write_cache_addr) || (addr > _write_cache_addr + _bd_program_size))) {
152      610                return _bd->read(b, addr, size);
153                     }
154
155    1702881          uint8_t *buf = static_cast<uint8_t *>(b);
156
157                     // Read logic: Split read to chunks, according to whether we cross the write cache
158   ✓✓ 5108645        while (size) {
```

```
159                 bd_size_t chunk;
160       1702882   bool read_from_bd = true;
161 ✓✓✗✓ 1702882   if (_write_cache_valid && addr < _write_cache_addr) {
162                     chunk = std::min(size, _write_cache_addr - addr);
163 ✓✓✓✗ 1702882   } else if (_write_cache_valid && (addr >= _write_cache_addr) && (addr < _write_cache_addr + _bd_program_size)) {
    ✓✓
164                     // One case we need to take our data from cache
165             2       chunk = std::min(size, _bd_program_size - addr % _bd_program_size);
166             2       memcpy(buf, _write_cache + addr % _bd_program_size, chunk);
167             2       read_from_bd = false;
168                 } else {
169       1702880       chunk = size;
170                 }
171
172                 // Now, in case we read from the BD, make sure we are aligned with its read size.
173                 // If not, use read buffer as a helper.
174     ✓✓ 1702882   if (read_from_bd) {
175       1702880       bd_size_t offs_in_read_buf = addr % _bd_read_size;
176                     int ret;
177 ✓✓✓✓ 1702880       if (offs_in_read_buf || (chunk < _bd_read_size)) {
178       1702879           chunk = std::min(chunk, _bd_read_size - offs_in_read_buf);
179     ✓✗ 1702879           ret = _bd->read(_read_buf, addr - offs_in_read_buf, _bd_read_size);
180       1702879           memcpy(buf, _read_buf + offs_in_read_buf, chunk);
181                     } else {
182             1           chunk = align_down(chunk, _bd_read_size);
183     ✓✗      1           ret = _bd->read(buf, addr, chunk);
184                     }
185     ✗✓ 1702880       if (ret) {
186                         return ret;
187                     }
188                 }
189
190       1702882   buf += chunk;
191       1702882   addr += chunk;
192       1702882   size -= chunk;
193             }
194
195       1702881   return 0;
196         }
197
198         62186 int BufferedBlockDevice::program(const void *b, bd_addr_t addr, bd_size_t size)
199         {
```

```
200  ✓✓  62186      if (!_is_initialized) {
201          1          return BD_ERROR_DEVICE_ERROR;
202                 }
203
204  ✗✓  62185      MBED_ASSERT(_write_cache);
205
206                 int ret;
207
208      62185      bd_addr_t aligned_addr = align_down(addr, _bd_program_size);
209
210      62185      const uint8_t *buf = static_cast <const uint8_t *>(b);
211
212                 // Need to flush if moved to another program unit
213  ✓✓  62185      if (aligned_addr != _write_cache_addr) {
214      51049          ret = flush();
215  ✗✓  51049          if (ret) {
216                         return ret;
217                     }
218                 }
219
220                 // Write logic: Keep data in cache as long as we don't reach the end of the program unit.
221                 // Otherwise, program to the underlying BD.
222  ✓✓ 186155      while (size) {
223      61985          _write_cache_addr = align_down(addr, _bd_program_size);
224      61985          bd_addr_t offs_in_buf = addr - _write_cache_addr;
225                     bd_size_t chunk;
226  ✓✓  61985          if (offs_in_buf) {
227      10937              chunk = std::min(_bd_program_size - offs_in_buf, size);
228  ✓✓  51048          } else if (size >= _bd_program_size) {
229      29794              chunk = align_down(size, _bd_program_size);
230                     } else {
231      21254              chunk = size;
232                     }
233
234                     const uint8_t *prog_buf;
235  ✓✓  61985          if (chunk < _bd_program_size) {
236                         // If cache not valid, and program doesn't cover an entire unit, it means we need to
237                         // read it from the underlying BD
238  ✓✓  32191              if (!_write_cache_valid) {
239  ✓✗  21255                  ret = _bd->read(_write_cache, _write_cache_addr, _bd_program_size);
240  ✗✓  21255                  if (ret) {
241                                 return ret;
```

```
242                         }
243                     }
244 32191              memcpy(_write_cache + offs_in_buf, buf, chunk);
245 32191              prog_buf = _write_cache;
246                 } else {
247 29794              prog_buf = buf;
248                 }
249
250                 // Only program if we reached the end of a program unit
251 ✓✓ 61985          if (!((offs_in_buf + chunk) % _bd_program_size)) {
252 ✓✗ 29795              ret = _bd->program(prog_buf, _write_cache_addr, std::max(chunk, _bd_program_size));
253 ✗✓ 29795              if (ret) {
254                         return ret;
255                     }
256 29795              invalidate_write_cache();
257 ✓✗ 29795              ret = _bd->sync();
258 ✗✓ 29795              if (ret) {
259                         return ret;
260                     }
261                 } else {
262 32190              _write_cache_valid = true;
263                 }
264
265 61985          buf += chunk;
266 61985          addr += chunk;
267 61985          size -= chunk;
268             }
269
270 62185      return 0;
271         }
272
273 16695  int BufferedBlockDevice::erase(bd_addr_t addr, bd_size_t size)
274         {
275 ✓✓ 16695      if (!_is_initialized) {
276     1             return BD_ERROR_DEVICE_ERROR;
277             }
278
279 ✓✓ 16694      if (!is_valid_erase(addr, size)) {
280     1             return BD_ERROR_DEVICE_ERROR;
281             }
282
283 ✓✗✓✓ 16693    if ((_write_cache_addr >= addr) && (_write_cache_addr <= addr + size)) {
```

```
284                 1            invalidate_write_cache();
285                              }
286             16693            return _bd->erase(addr, size);
287                              }
288                              
289                 1            int BufferedBlockDevice::trim(bd_addr_t addr, bd_size_t size)
290                              {
291   ✓✗            1            if (!_is_initialized) {
292                 1                return BD_ERROR_DEVICE_ERROR;
293                              }
294                              
295                              if (!is_valid_erase(addr, size)) {
296                                  return BD_ERROR_DEVICE_ERROR;
297                              }
298                              
299                              if ((_write_cache_addr >= addr) && (_write_cache_addr <= addr + size)) {
300                                  invalidate_write_cache();
301                              }
302                              return _bd->trim(addr, size);
303                              }
304                              
305           3406985            bd_size_t BufferedBlockDevice::get_read_size() const
306                              {
307           3406985            return 1;
308                              }
309                              
310                 1            bd_size_t BufferedBlockDevice::get_program_size() const
311                              {
312                 1            return 1;
313                              }
314                              
315                 2            bd_size_t BufferedBlockDevice::get_erase_size() const
316                              {
317   ✓✓            2            if (!_is_initialized) {
318                 1                return 0;
319                              }
320                              
321                 1            return _bd->get_erase_size();
322                              }
323                              
324             61971            bd_size_t BufferedBlockDevice::get_erase_size(bd_addr_t addr) const
325                              {
```

```
326     ✓✓  61971     if (!_is_initialized) {
327             1         return 0;
328                   }
329
330         61970     return _bd->get_erase_size(addr);
331                 }
332
333         16694 int BufferedBlockDevice::get_erase_value() const
334                 {
335     ✓✓  16694     if (!_is_initialized) {
336             1         return BD_ERROR_DEVICE_ERROR;
337                   }
338
339         16693     return _bd->get_erase_value();
340                 }
341
342       1720187 bd_size_t BufferedBlockDevice::size() const
343                 {
344     ✓✓1720187     if (!_is_initialized) {
345             1         return 0;
346                   }
347
348       1720186     return _bd_size;
349                 }
350
351             1 const char *BufferedBlockDevice::get_type() const
352                 {
353             1     return _bd->get_type();
354                 }
355
356                 } // namespace mbed
```