A Practical Guide to Automated Testing

May 30th 2018

Andrew Jeffery
andrewrj@au1.ibm.com

Assumptions

- C or C++
- Autotools-based project
- lcov
  - Ubuntu: `sudo apt install lcov`
- Use of Google Test and Google Mock libraries

phosphor-led-sysfs

https://gerrit.openbmc-project.xyz/#/q/project:openbmc/phosphor-led-sysfs+topic:testing

Automated testing...

- Is automated code review
  - Less work to do once the tests are in place
- Provides confidence code is working as expected
- Enforces expected behaviour in the face of change

But only for the execution paths that your tests touch!

The first step

https://gerrit.openbmc-project.xyz/#/c/10827/

```
diff --git a/configure.ac b/configure.ac
index 6a08be339e16..211e4efe1b02 100644
--- a/configure.ac
+++ b/configure.ac
@@ -29,6 +29,8 @@ PKG_CHECK_MODULES([PHOSPHOR_DBUS_INTERFACES], [phosphor-dbus-interfaces],, [AC_M
 AX_PTHREAD([GTEST_CPPFLAGS="-DGTEST_HAS_PTHREAD=1"],[GTEST_CPPFLAGS="-DGTEST_HAS_PTHREAD=0"])
 AC_SUBST(GTEST_CPPFLAGS)

+AX_CODE_COVERAGE
+
 AC_ARG_VAR(BUSNAME, [The Dbus busname to own])
 AS_IF([test "x$BUSNAME" == "x"], [BUSNAME="xyz.openbmc_project.LED.Controller"])
 AC_DEFINE_UNQUOTED([BUSNAME], ["$BUSNAME"], [The Dbus busname to own])
```

The first step

https://gerrit.openbmc-project.xyz/#/c/10827/

```
diff --git a/Makefile.am b/Makefile.am
index 3c42e07184d7..a5e625fc9160 100644
--- a/Makefile.am
+++ b/Makefile.am
@@ -9,3 +9,10 @@ phosphor_ledcontroller_LDFLAGS = $(SDBUSPLUS_LIBS) \
                                 $(PHOSPHOR_DBUS_INTERFACES_LIBS)
 phosphor_ledcontroller_CFLAGS =  $(SDBUSPLUS_CFLAGS) \
                                 $(PHOSPHOR_DBUS_INTERFACES_CFLAGS)
+
+@CODE_COVERAGE_RULES@
+
+check_PROGRAMS =
+XFAIL_TESTS =
+
+TESTS = $(check_PROGRAMS)
```

The first result

```
$ ./configure –enable-code-coverage && make check-code-coverage
...
============================================================================
Testsuite summary for phosphor-led-sysfs 1.0
============================================================================
# TOTAL: 0
# PASS:  0
# SKIP:  0
# XFAIL: 0
# FAIL:  0
# XPASS: 0
# ERROR: 0
============================================================================
make[3]: Leaving directory '/home/andrew/src/openbmc/phosphor-led-sysfs'
make[2]: Leaving directory '/home/andrew/src/openbmc/phosphor-led-sysfs'
make[1]: Leaving directory '/home/andrew/src/openbmc/phosphor-led-sysfs'
make[1]: Entering directory '/home/andrew/src/openbmc/phosphor-led-sysfs'
  LCOV   --capture phosphor-led-sysfs-1.0-coverage.info
geninfo: WARNING: no .gcda files found in . - skipping!
  LCOV   --remove /tmp/*
lcov: ERROR: no valid records found in tracefile phosphor-led-sysfs-1.0-coverage.info.tmp
Makefile:1329: recipe for target 'code-coverage-capture' failed
make[1]: *** [code-coverage-capture] Error 255
make[1]: Leaving directory '/home/andrew/src/openbmc/phosphor-led-sysfs'
Makefile:1323: recipe for target 'check-code-coverage' failed
make: *** [check-code-coverage] Error 2
```

phosphor-led-sysfs

```
$ git ls-files | grep '\.[ch]pp'
argument.cpp
argument.hpp
controller.cpp
physical.cpp
physical.hpp
$
```

phosphor-led-sysfs

- **argument.[ch]pp**: Argument parsing
- **controller.cpp**: main(), orchestrates everything
- **physical.[ch]pp**: Testable business logic!
  - Converts DBus API to kernel LED sysfs ABI

Testing physical.cpp

https://gerrit.openbmc-project.xyz/#/c/10829/

```
diff --git a/Makefile.am b/Makefile.am
index a5e625fc9160..429415b54688 100644
--- a/Makefile.am
+++ b/Makefile.am
@@ -15,4 +15,6 @@ phosphor_ledcontroller_CFLAGS =  $(SDBUSPLUS_CFLAGS) \
 check_PROGRAMS =
 XFAIL_TESTS =

+include test/Makefile.am.include
+
 TESTS = $(check_PROGRAMS)
```

Testing physical.cpp

https://gerrit.openbmc-project.xyz/#/c/10829/

```
diff --git a/test/Makefile.am.include b/test/Makefile.am.include
new file mode 100644
index 000000000000..32b97f79c6ef
--- /dev/null
+++ b/test/Makefile.am.include
@@ -0,0 +1,13 @@
+GTEST_LIBS = -lgtest -lgtest_main -lgmock
+AM_CPPFLAGS = $(CODE_COVERAGE_CPPFLAGS) $(PTHREAD_CPPFLAGS) $(SDBUSPLUS_CPPFLAGS) $(PHOSPHOR_DBUS_INTERFACES_CPPFLAGS)
+AM_CFLAGS = $(CODE_COVERAGE_CFLAGS) $(PTHREAD_CFLAGS) $(SDBUSPLUS_CFLAGS) $(PHOSPHOR_DBUS_INTERFACES_CFLAGS)
+AM_CXXFLAGS = $(CODE_COVERAGE_CXXFLAGS) $(PTHREAD_CXXFLAGS) $(SDBUSPLUS_CXXFLAGS) $(PHOSPHOR_DBUS_INTERFACES_CXXFLAGS)
+AM_LDFLAGS = $(CODE_COVERAGE_LIBS) $(SDBUSPLUS_LIBS) $(PHOSPHOR_DBUS_INTERFACES_LIBS) $(GTEST_LIBS) $(PTHREAD_CFLAGS)
+
+test_physical_SOURCES = physical.cpp %reldir%/physical.cpp
+test_physical_CPPFLAGS = $(AM_CPPFLAGS) $(GTEST_CPPFLAGS)
+
+check_PROGRAMS += %reldir%/physical
```

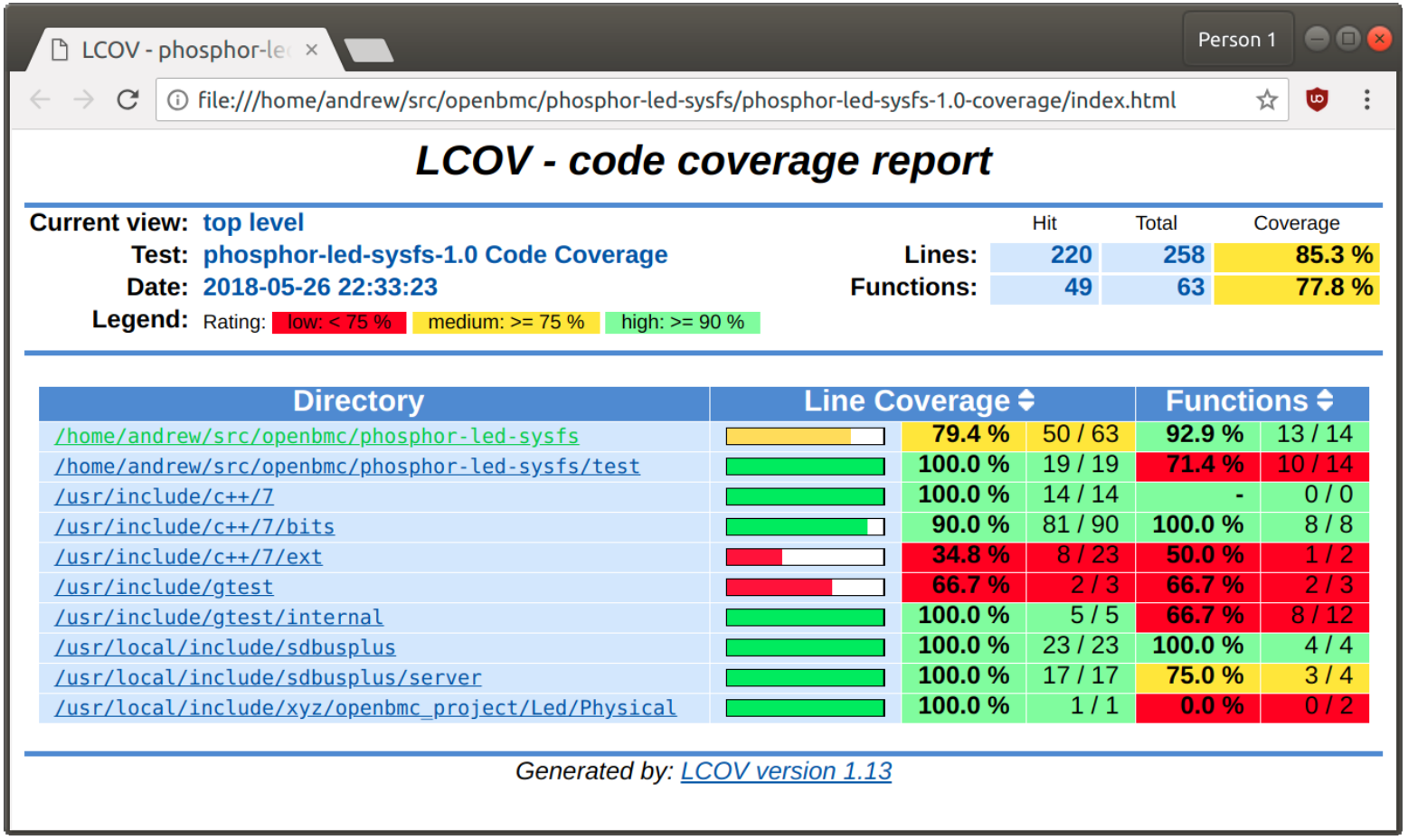
Testing physical.[ch]pp

```
diff --git a/test/physical.cpp b/test/physical.cpp
new file mode 100644
index 000000000000..511bbfa4f70e
--- /dev/null
+++ b/test/physical.cpp
@@ -0,0 +1,36 @@
+#include <gtest/gtest.h>
+#include <sdbusplus/bus.hpp>
+
+#include "physical.hpp"
+
+constexpr auto LED_OBJ = "/foo/bar/led";
+constexpr auto LED_SYSFS = "/sys/class/leds/test";
+
+using Action = sdbusplus::xyz::openbmc_project::Led::server::Physical::Action;
+
+TEST(Physical, ctor)
+{
+    sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
+    phosphor::led::Physical led(bus, LED_OBJ, LED_SYSFS);
+}
+
+TEST(Physical, off)
+{
+    sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
+    phosphor::led::Physical led(bus, LED_OBJ, LED_SYSFS);
+    led.state(Action::Off);
+}
+
+TEST(Physical, on)
+{
+    sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
+    phosphor::led::Physical led(bus, LED_OBJ, LED_SYSFS);
+    led.state(Action::On);
+}
+
+TEST(Physical, blink)
+{
+    sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
+    phosphor::led::Physical led(bus, LED_OBJ, LED_SYSFS);
+    led.state(Action::Blink);
+}
```

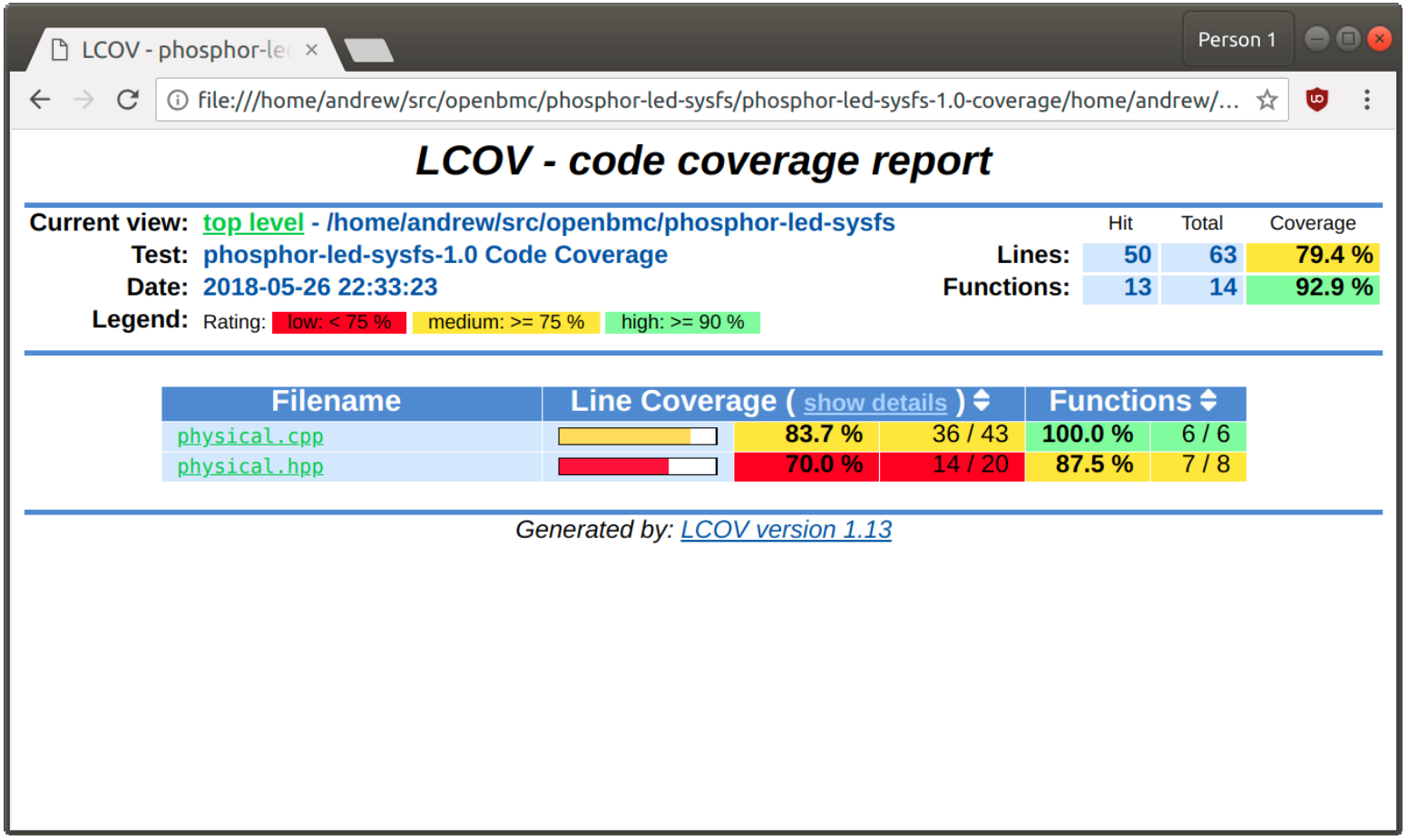
The second result

```
$ ./configure –enable-code-coverage && make check-code-coverage
...
============================================================================
Testsuite summary for phosphor-led-sysfs 1.0
============================================================================
# TOTAL: 1
# PASS:  1
# SKIP:  0
# XFAIL: 0
# FAIL:  0
# XPASS: 0
# ERROR: 0
============================================================================
make[3]: Leaving directory '/home/andrew/src/openbmc/phosphor-led-sysfs'
make[2]: Leaving directory '/home/andrew/src/openbmc/phosphor-led-sysfs'
make[1]: Leaving directory '/home/andrew/src/openbmc/phosphor-led-sysfs'
make[1]: Entering directory '/home/andrew/src/openbmc/phosphor-led-sysfs'
  LCOV   --capture phosphor-led-sysfs-1.0-coverage.info
  LCOV   --remove /tmp/*
  GEN    phosphor-led-sysfs-1.0-coverage
file:///home/andrew/src/openbmc/phosphor-led-sysfs/phosphor-led-sysfs-1.0-coverage/index.html
make[1]: Leaving directory '/home/andrew/src/openbmc/phosphor-led-sysfs'
```

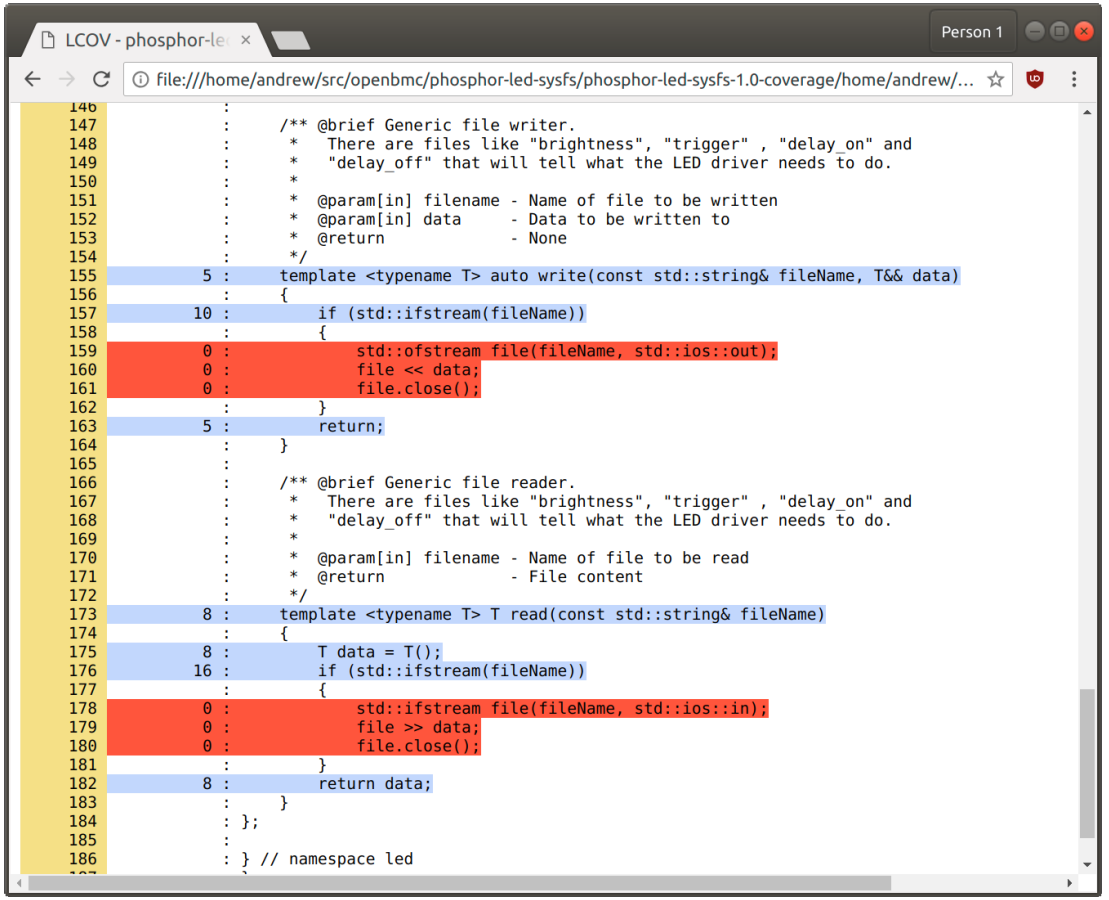
The second result



The second result

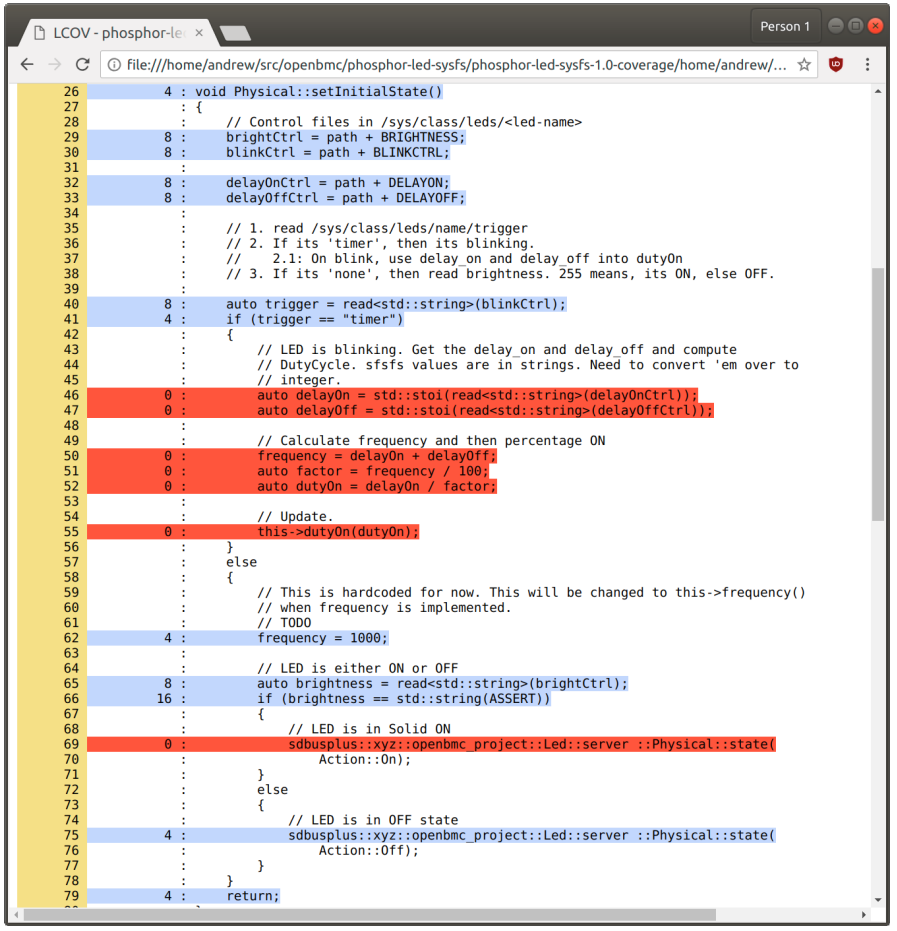


What we missed: physical.hpp




```
146          :
147          :     /** @brief Generic file writer.
148          :      *   There are files like "brightness", "trigger" , "delay_on" and
149          :      *   "delay_off" that will tell what the LED driver needs to do.
150          :      *
151          :      *  @param[in] filename - Name of file to be written
152          :      *  @param[in] data     - Data to be written to
153          :      *  @return             - None
154          :      */
155        5 :     template <typename T> auto write(const std::string& fileName, T&& data)
156          :     {
157       10 :         if (std::ifstream(fileName))
158          :         {
159        0 :             std::ofstream file(fileName, std::ios::out);
160        0 :             file << data;
161        0 :             file.close();
162          :         }
163        5 :         return;
164          :     }
165          :
166          :     /** @brief Generic file reader.
167          :      *   There are files like "brightness", "trigger" , "delay_on" and
168          :      *   "delay_off" that will tell what the LED driver needs to do.
169          :      *
170          :      *  @param[in] filename - Name of file to be read
171          :      *  @return             - File content
172          :      */
173        8 :     template <typename T> T read(const std::string& fileName)
174          :     {
175        8 :         T data = T();
176       16 :         if (std::ifstream(fileName))
177          :         {
178        0 :             std::ifstream file(fileName, std::ios::in);
179        0 :             file >> data;
180        0 :             file.close();
181          :         }
182        8 :         return data;
183          :     }
184          : };
185          :
186          : } // namespace led
```

What we missed: physical.cpp




```
26        4 : void Physical::setInitialState()
27          : {
28          :     // Control files in /sys/class/leds/<led-name>
29        8 :     brightCtrl = path + BRIGHTNESS;
30        8 :     blinkCtrl = path + BLINKCTRL;
31          :
32        8 :     delayOnCtrl = path + DELAYON;
33        8 :     delayOffCtrl = path + DELAYOFF;
34          :
35          :     // 1. read /sys/class/leds/name/trigger
36          :     // 2. If its 'timer', then its blinking.
37          :     //    2.1: On blink, use delay_on and delay_off into dutyOn
38          :     // 3. If its 'none', then read brightness. 255 means, its ON, else OFF.
39          :
40        8 :     auto trigger = read<std::string>(blinkCtrl);
41        4 :     if (trigger == "timer")
42          :     {
43          :         // LED is blinking. Get the delay_on and delay_off and compute
44          :         // DutyCycle. sfsfs values are in strings. Need to convert 'em over to
45          :         // integer.
46        0 :         auto delayOn = std::stoi(read<std::string>(delayOnCtrl));
47        0 :         auto delayOff = std::stoi(read<std::string>(delayOffCtrl));
48          :
49          :         // Calculate frequency and then percentage ON
50        0 :         frequency = delayOn + delayOff;
51        0 :         auto factor = frequency / 100;
52        0 :         auto dutyOn = delayOn / factor;
53          :
54          :         // Update.
55        0 :         this->dutyOn(dutyOn);
56          :     }
57          :     else
58          :     {
59          :         // This is hardcoded for now. This will be changed to this->frequency()
60          :         // when frequency is implemented.
61          :         // TODO
62        4 :         frequency = 1000;
63          :
64          :         // LED is either ON or OFF
65        8 :         auto brightness = read<std::string>(brightCtrl);
66       16 :         if (brightness == std::string(ASSERT))
67          :         {
68          :             // LED is in Solid ON
69        0 :             sdbusplus::xyz::openbmc_project::Led::server ::Physical::state(
70          :                 Action::On);
71          :         }
72          :         else
73          :         {
74          :             // LED is in OFF state
75        4 :             sdbusplus::xyz::openbmc_project::Led::server ::Physical::state(
76          :                 Action::Off);
77          :         }
78          :     }
79        4 :     return;
```

The unhelpful design of Physical

1) read() and write() are non-virtual functions
2) read() is called by the constructor
3) read() and write() are template functions
4) There is a better choice of abstraction

sysfs.[ch]pp

https://gerrit.openbmc-project.xyz/#/c/10830/

```
diff --git a/sysfs.hpp b/sysfs.hpp
new file mode 100644
index 000000000000..e88cfd09465b
--- /dev/null
+++ b/sysfs.hpp
@@ -0,0 +1,39 @@
+namespace phosphor {
+namespace led {
+class SysfsLed
+{
+  public:
+    SysfsLed(std::experimental::filesystem::path&& root) : root(root) { }
+    virtual ~SysfsLed() { }
+    virtual unsigned long getBrightness();
+    virtual void setBrightness(unsigned long value);
+    virtual unsigned long getMaxBrightness();
+    virtual std::string getTrigger();
+    virtual void setTrigger(std::string trigger);
+    virtual unsigned long getDelayOn();
+    virtual void setDelayOn(unsigned long ms);
+    virtual unsigned long getDelayOff();
+    virtual void setDelayOff(unsigned long ms);
+  protected:
+    std::experimental::filesystem::path root;
+};
+}
+}
```

```
diff --git a/sysfs.cpp b/sysfs.cpp
new file mode 100644
index 000000000000..89b85e125c08
--- /dev/null
+++ b/sysfs.cpp
@@ -0,0 +1,86 @@
+namespace fs = std::experimental::filesystem;
+namespace phosphor {
+namespace led {
+template <typename T> T get_sysfs_attr(fs::path&& path);
+template <typename T> void set_sysfs_attr(fs::path&& path, T& value)
+
+unsigned long SysfsLed::getBrightness()
+{
+    return get_sysfs_attr<unsigned long>(root / BRIGHTNESS);
+}
+
+void SysfsLed::setBrightness(unsigned long brightness)
+{
+    set_sysfs_attr<unsigned long>(root / BRIGHTNESS, brightness);
+}
+
+unsigned long SysfsLed::getMaxBrightness()
+{
+    return get_sysfs_attr<unsigned long>(root / MAX_BRIGHTNESS);
+}
...
```

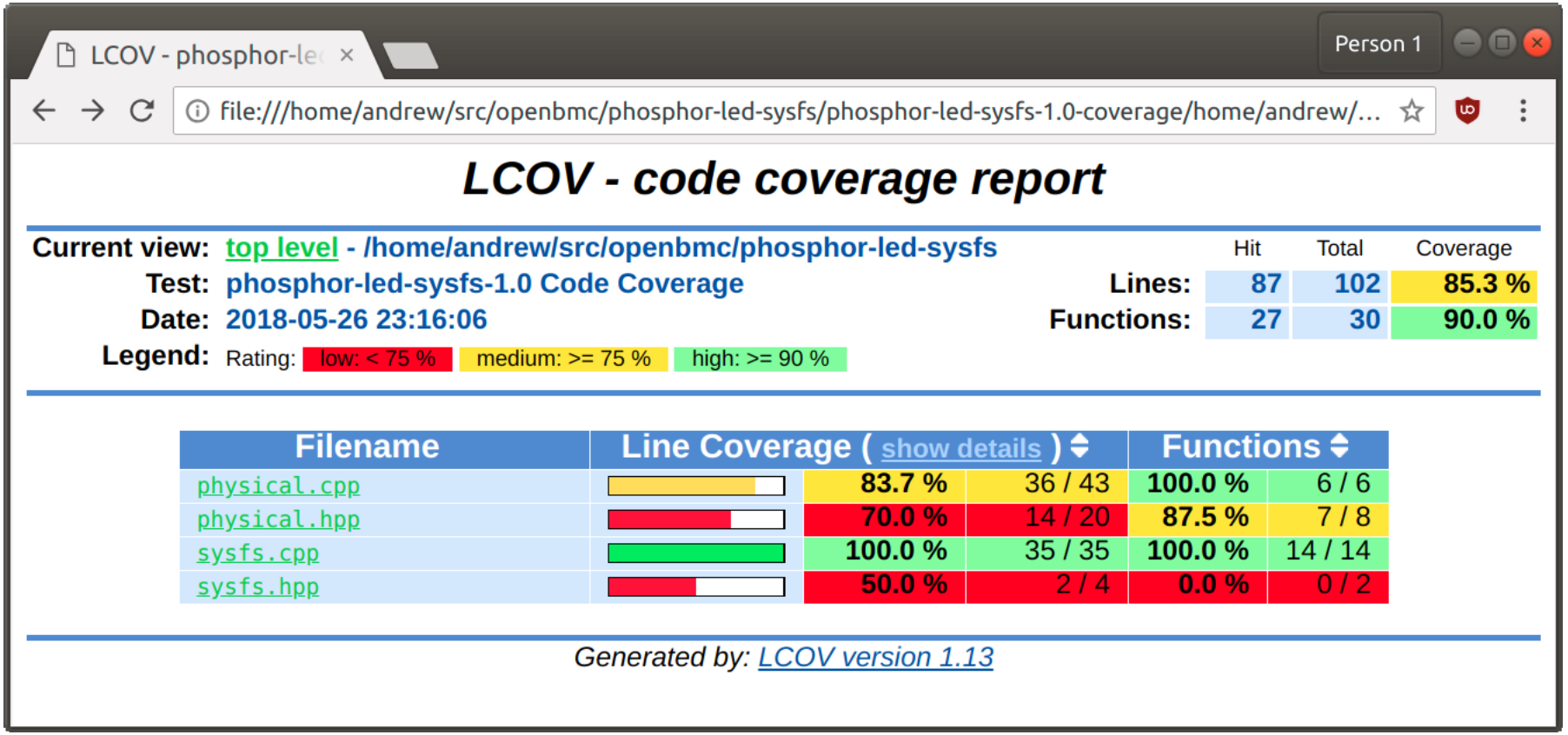
test/sysfs.cpp

https://gerrit.openbmc-project.xyz/#/c/10830/

```
+class FakeSysfsLed : public phosphor::led::SysfsLed
+{
+  public:
+    static FakeSysfsLed create()
+    {
+        const char* const tmplt = "/tmp/FakeSysfsLed.XXXXXX";
+        char buffer[MAXPATHLEN] = {0};
+
+        strncpy(buffer, tmplt, strlen(tmplt));
+        char* dir = mkdtemp(buffer);
+        if (!dir)
+            throw std::system_error(errno, std::system_category());
+
+        return FakeSysfsLed(fs::path(dir));
+    }
+
+    ~FakeSysfsLed()
+    {
+        fs::remove_all(root);
+    }
+
+  private:
+    FakeSysfsLed(fs::path&& path) : SysfsLed(std::move(path))
+    {
+        std::string attrs[4] = {BRIGHTNESS, TRIGGER, DELAY_ON, DELAY_OFF};
+        for (auto attr : attrs)
+        {
+            fs::path p = root / attr;
+            std::ofstream f(p, std::ios::out);
+            f.exceptions(f.failbit);
+        }
+
+        fs::path p = root / MAX_BRIGHTNESS;
+        std::ofstream f(p, std::ios::out);
+        f.exceptions(f.failbit);
+        f << MAX_BRIGHTNESS_VAL;
+    }
+};
```

```
+TEST(Sysfs, getBrightness)
+{
+    constexpr auto brightness = 127;
+    FakeSysfsLed fsl = FakeSysfsLed::create();
+
+    fsl.setBrightness(brightness);
+    ASSERT_EQ(brightness, fsl.getBrightness());
+}
+
+TEST(Sysfs, getMaxBrightness)
+{
+    FakeSysfsLed fsl = FakeSysfsLed::create();
+
+    ASSERT_EQ(MAX_BRIGHTNESS_VAL, fsl.getMaxBrightness());
+}
+
+TEST(Sysfs, getTrigger)
+{
+    constexpr auto trigger = "none";
+    FakeSysfsLed fsl = FakeSysfsLed::create();
+
+    fsl.setTrigger(trigger);
+    ASSERT_EQ(trigger, fsl.getTrigger());
+}
+
+TEST(Sysfs, getDelayOn)
+{
+    constexpr auto delayOn = 250;
+    FakeSysfsLed fsl = FakeSysfsLed::create();
+
+    fsl.setDelayOn(delayOn);
+    ASSERT_EQ(delayOn, fsl.getDelayOn());
+}
+
+TEST(Sysfs, getDelayOff)
+{
+    constexpr auto delayOff = 750;
+    FakeSysfsLed fsl = FakeSysfsLed::create();
+
+    fsl.setDelayOff(delayOff);
+    ASSERT_EQ(delayOff, fsl.getDelayOff());
+}
```

The third result



Integrate SysfsLed into Physical

https://gerrit.openbmc-project.xyz/#/c/10831/

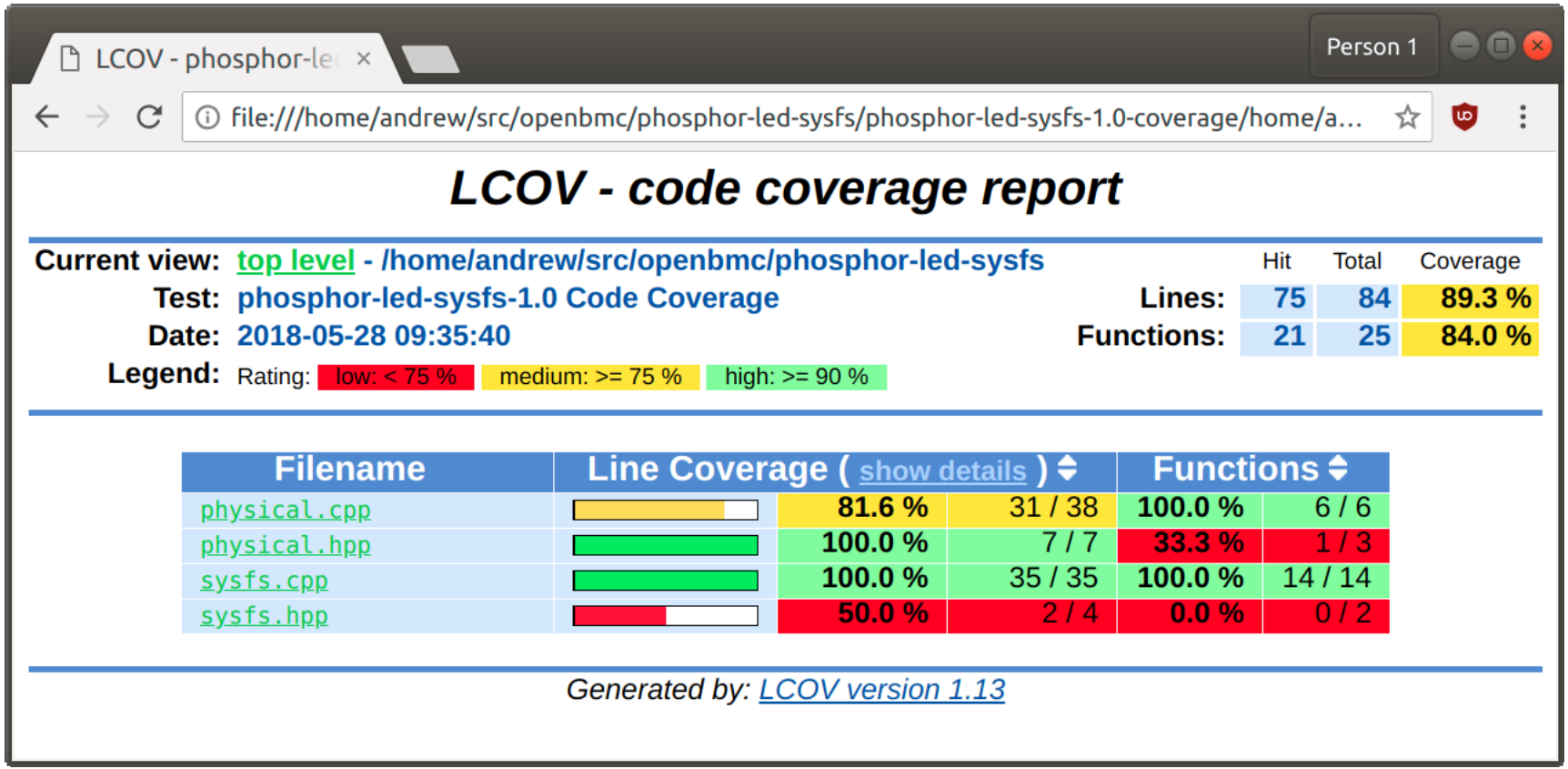
```
@@ -70,12 +42,12 @@ class Physical : public sdbusplus::server::object::object<
     * @param[in] ledPath   - sysfs path where this LED is exported
     */
    Physical(sdbusplus::bus::bus& bus, const std::string& objPath,
-            const std::string& ledPath) :
+            SysfsLed& led) :

        sdbusplus::server::object::object<
            sdbusplus::xyz::openbmc_project::Led::server::Physical>(
            bus, objPath.c_str(), true),
-       path(ledPath)
+       led(led)
    {
        // Suppose this is getting launched as part of BMC reboot, then we
        // need to save what the micro-controller currently has.
@@ -96,25 +68,13 @@ class Physical : public sdbusplus::server::object::object<
    /** @brief File system location where this LED is exposed
     *   Typically /sys/class/leds/<Led-Name>
     */
-   std::string path;
+   SysfsLed& led;

    /** @brief Frequency range that the LED can operate on.
     *  Will be removed when frequency is put into interface
     */
    uint32_t frequency;
```

```
-
-/** @brief Generic file writer.
- *   There are files like "brightness", "trigger" , "delay_on" and
- *   "delay_off" that will tell what the LED driver needs to do.
- *
- *  @param[in] filename - Name of file to be written
- *  @param[in] data     - Data to be written to
- *  @return             - None
- */
-template <typename T> auto write(const std::string& fileName, T&& data)
-{
-    if (std::ifstream(fileName))
-    {
-        std::ofstream file(fileName, std::ios::out);
-        file << data;
-        file.close();
-    }
-    return;
-}
-
-/** @brief Generic file reader.
- *   There are files like "brightness", "trigger" , "delay_on" and
- *   "delay_off" that will tell what the LED driver needs to do.
- *
- *  @param[in] filename - Name of file to be read
- *  @return             - File content
- */
-template <typename T> T read(const std::string& fileName)
-{
-    T data = T();
-    if (std::ifstream(fileName))
-    {
-        std::ifstream file(fileName, std::ios::in);
-        file >> data;
-        file.close();
-    }
-    return data;
-}
```

The fourth result



Mocking SysfsLed

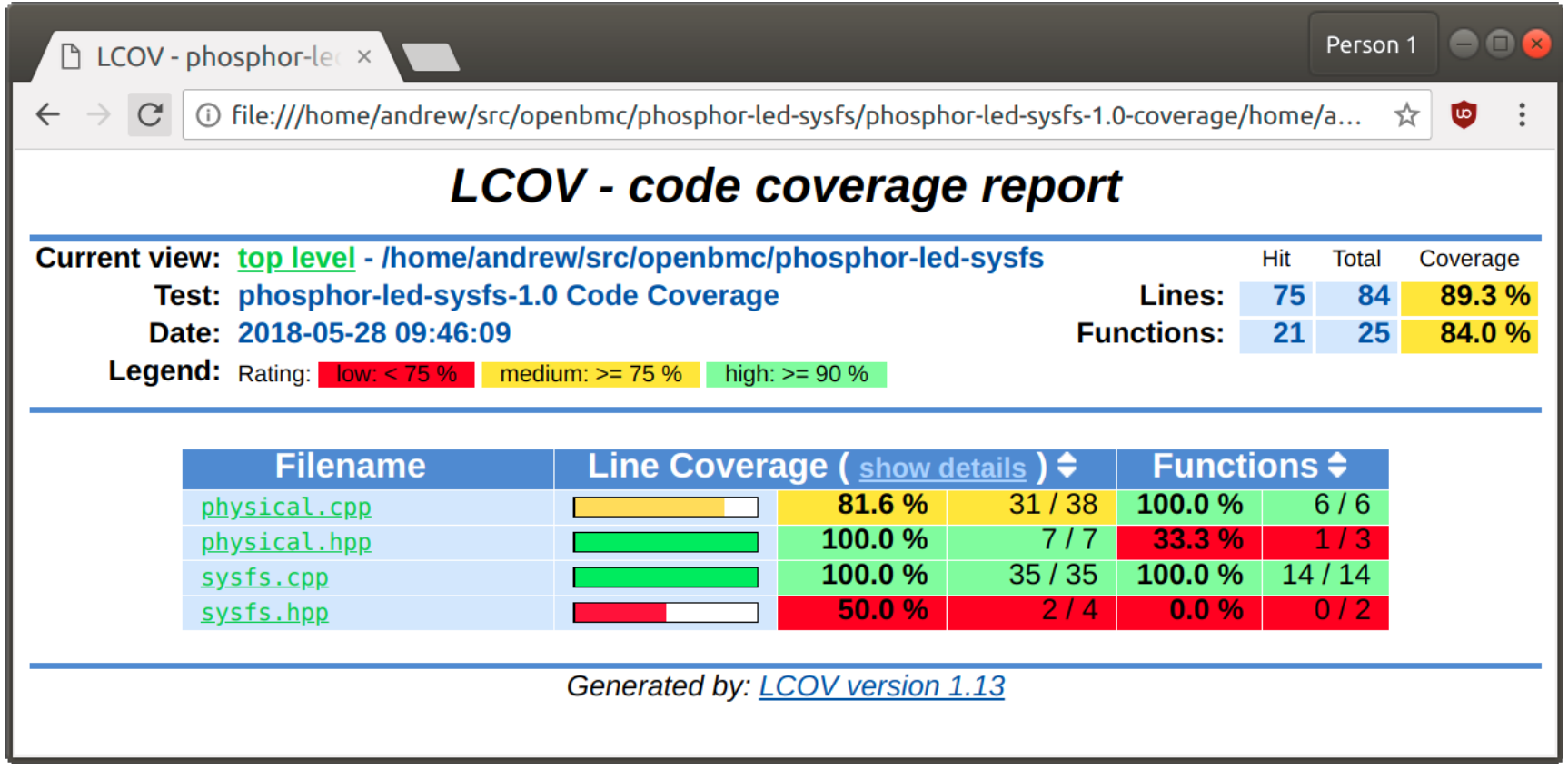
https://gerrit.openbmc-project.xyz/#/c/10832/

```
diff --git a/test/physical.cpp b/test/physical.cpp
index fd1157e9a115..eab8ac9b03dc 100644
--- a/test/physical.cpp
+++ b/test/physical.cpp
@@ -1,25 +1,87 @@
 #include <gtest/gtest.h>
+#include <gmock/gmock.h>
...
+class MockLed : public phosphor::led::SysfsLed
+{
+  public:
+    /* Use a no-args ctor here to avoid headaches with {Nice,Strict}Mock */
+    MockLed() : SysfsLed(…) { }
+    virtual ~MockLed() { ... }
+
+    MOCK_METHOD0(getBrightness, unsigned long());
+    MOCK_METHOD1(setBrightness, void(unsigned long value));
+    MOCK_METHOD0(getMaxBrightness, unsigned long());
+    MOCK_METHOD0(getTrigger, std::string());
+    MOCK_METHOD1(setTrigger, void(std::string trigger));
+    MOCK_METHOD0(getDelayOn, unsigned long());
+    MOCK_METHOD1(setDelayOn, void(unsigned long ms));
+    MOCK_METHOD0(getDelayOff, unsigned long());
+    MOCK_METHOD1(setDelayOff, void(unsigned long ms));
+};
```

```
+using ::testing::NiceMock;
+using ::testing::Return;
+
 TEST(Physical, ctor)
 {
     sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
-    phosphor::led::SysfsLed led = phosphor::led::SysfsLed(fs::path(LED_SYSFS));
+    /* NiceMock ignores calls to methods with no expectations defined */
+    NiceMock<MockLed> led;
+    ON_CALL(led, getTrigger()).WillByDefault(Return("none"));
     phosphor::led::Physical phy(bus, LED_OBJ, led);
 }

 TEST(Physical, off)
 {
     sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
-    phosphor::led::SysfsLed led = phosphor::led::SysfsLed(fs::path(LED_SYSFS));
+    NiceMock<MockLed> led;
+    ON_CALL(led, getTrigger()).WillByDefault(Return("none"));
     phosphor::led::Physical phy(bus, LED_OBJ, led);
     phy.state(Action::Off);
 }
@@ -27,7 +89,8 @@ TEST(Physical, off)
 TEST(Physical, on)
 {
     sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
-    phosphor::led::SysfsLed led = phosphor::led::SysfsLed(fs::path(LED_SYSFS));
+    NiceMock<MockLed> led;
+    ON_CALL(led, getTrigger()).WillByDefault(Return("none"));
     phosphor::led::Physical phy(bus, LED_OBJ, led);
     phy.state(Action::On);
 }
@@ -35,7 +98,9 @@ TEST(Physical, on)
 TEST(Physical, blink)
 {
     sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
-    phosphor::led::SysfsLed led = phosphor::led::SysfsLed(fs::path(LED_SYSFS));
+    NiceMock<MockLed> led;
+    ON_CALL(led, getTrigger()).WillByDefault(Return("none"));
+    ON_CALL(led, getDelayOn()).WillByDefault(Return(500));
     phosphor::led::Physical phy(bus, LED_OBJ, led);
     phy.state(Action::Blink);
 }
```

The fifth result



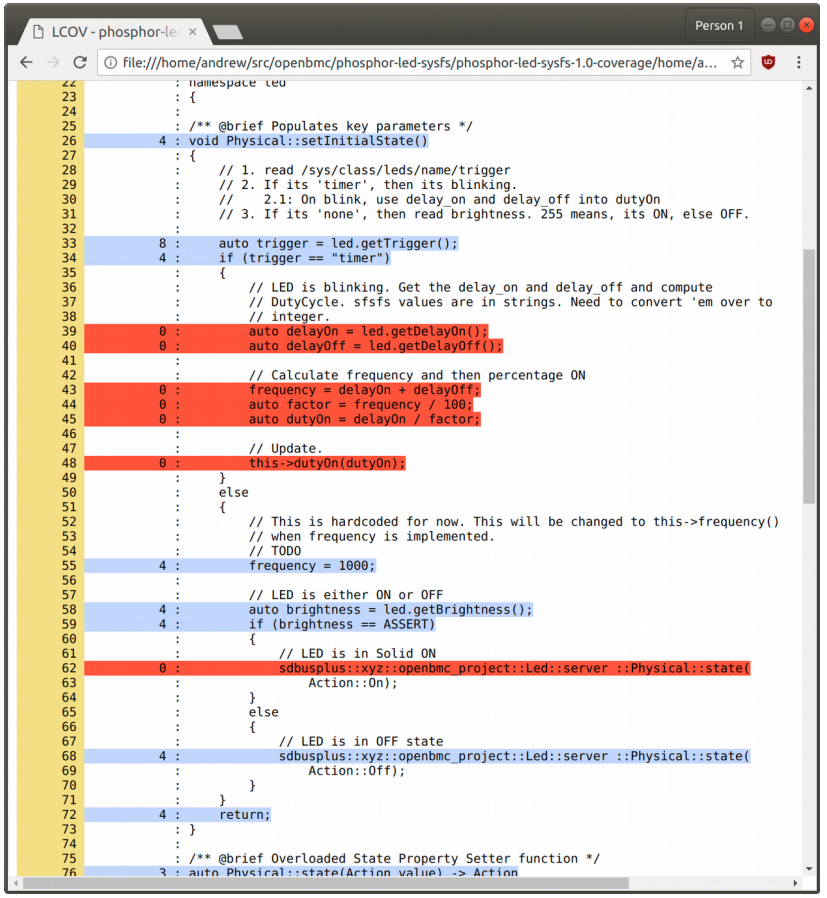
LCOV - code coverage report

		Hit	Total	Coverage
Current view:	top level - /home/andrew/src/openbmc/phosphor-led-sysfs			
Test:	phosphor-led-sysfs-1.0 Code Coverage	**Lines:**	75	84	89.3 %
Date:	2018-05-28 09:46:09	**Functions:**	21	25	84.0 %
Legend:	Rating: low: < 75 % medium: >= 75 % high: >= 90 %			

Filename	Line Coverage (show details)		Functions	
physical.cpp	81.6 %	31 / 38	100.0 %	6 / 6
physical.hpp	100.0 %	7 / 7	33.3 %	1 / 3
sysfs.cpp	100.0 %	35 / 35	100.0 %	14 / 14
sysfs.hpp	50.0 %	2 / 4	0.0 %	0 / 2

Generated by: LCOV version 1.13

The un-hit paths in Physical.cpp



Hitting ‘trigger == “timer”’

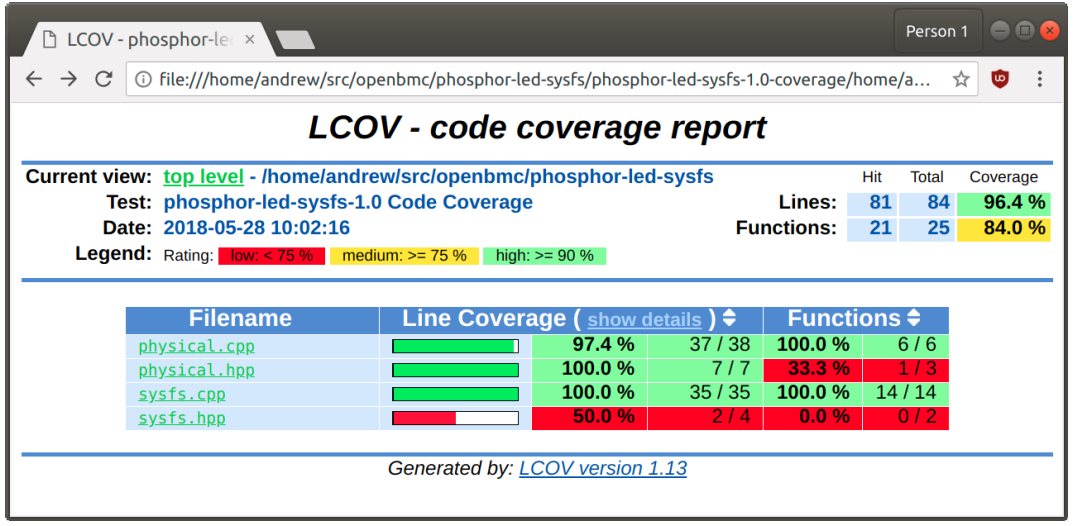
https://gerrit.openbmc-project.xyz/#/c/10833/

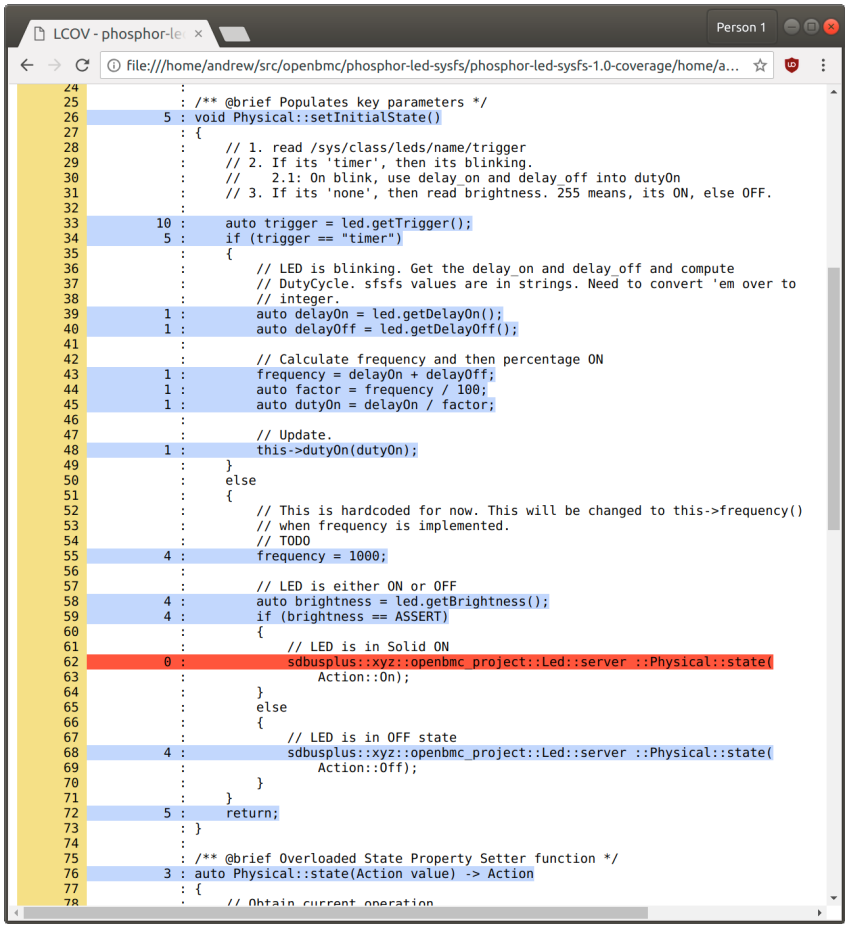
```
diff --git a/test/physical.cpp b/test/physical.cpp
index eab8ac9b03dc..a853dcd748aa 100644
--- a/test/physical.cpp
+++ b/test/physical.cpp
@@ -68,7 +68,7 @@ using ::testing::NiceMock;
 using ::testing::Return;
 using ::testing::Throw;

-TEST(Physical, ctor)
+TEST(Physical, ctor_none_trigger)
 {
     sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
     /* NiceMock ignores calls to methods with no expectations
defined */
@@ -77,6 +77,16 @@ TEST(Physical, ctor)
     phosphor::led::Physical phy(bus, LED_OBJ, led);
 }

+TEST(Physical, ctor_timer_trigger)
+{
+    sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
+    NiceMock<MockLed> led;
+    EXPECT_CALL(led, getTrigger()).WillOnce(Return("timer"));
+    EXPECT_CALL(led, getDelayOn()).WillOnce(Return(500));
+    EXPECT_CALL(led, getDelayOff()).WillOnce(Return(500));
+    phosphor::led::Physical phy(bus, LED_OBJ, led);
+}
+
```

The sixth result



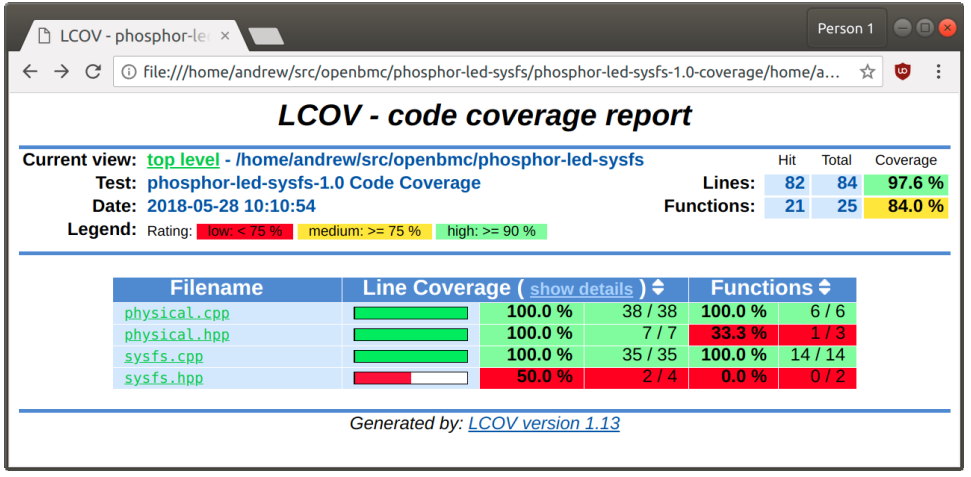


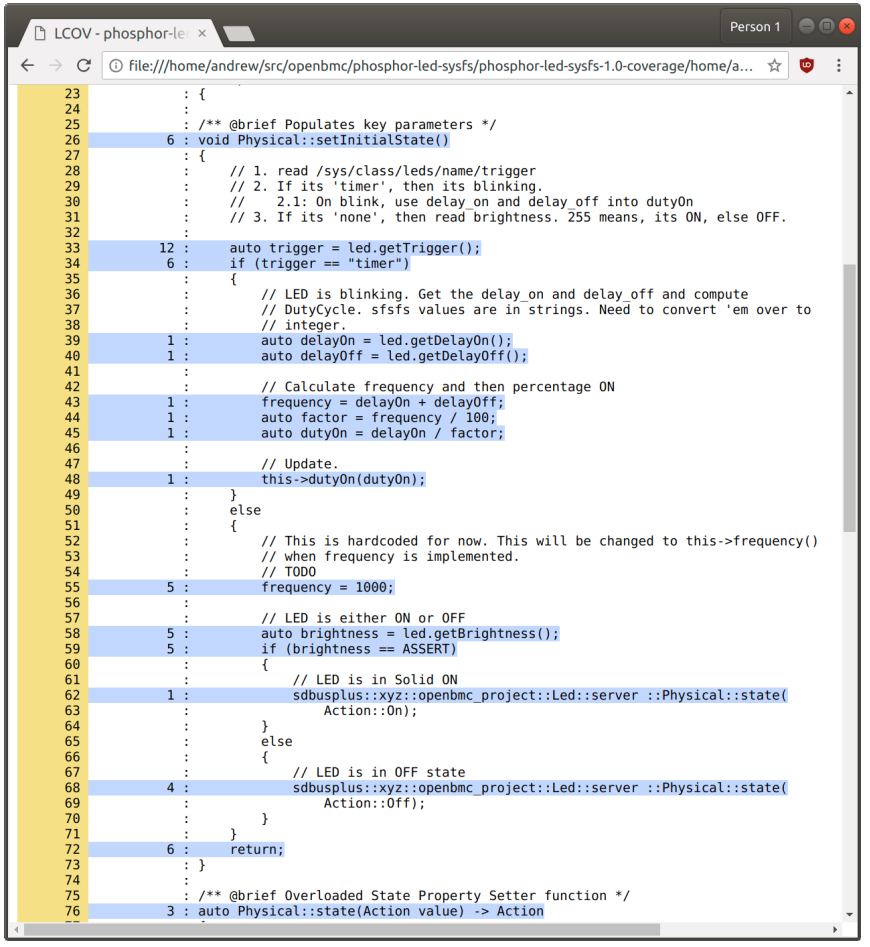
Hitting ‘brightness == ASSERT’

https://gerrit.openbmc-project.xyz/#/c/10834/

```
diff --git a/test/physical.cpp b/test/physical.cpp
index a853dcd748aa..9bb8424bee8f 100644
--- a/test/physical.cpp
+++ b/test/physical.cpp
@@ -114,3 +114,13 @@ TEST(Physical, blink)
     phosphor::led::Physical phy(bus, LED_OBJ, led);
     phy.state(Action::Blink);
 }
+
+TEST(Physical, ctor_none_trigger_asserted_brightness)
+{
+    sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
+    NiceMock<MockLed> led;
+    EXPECT_CALL(led, getTrigger()).WillRepeatedly(Return("none"));
+    constexpr auto val = phosphor::led::ASSERT;
+    EXPECT_CALL(led, getBrightness()).WillRepeatedly(Return(val));
+    phosphor::led::Physical phy(bus, LED_OBJ, led);
+}
```

The seventh result





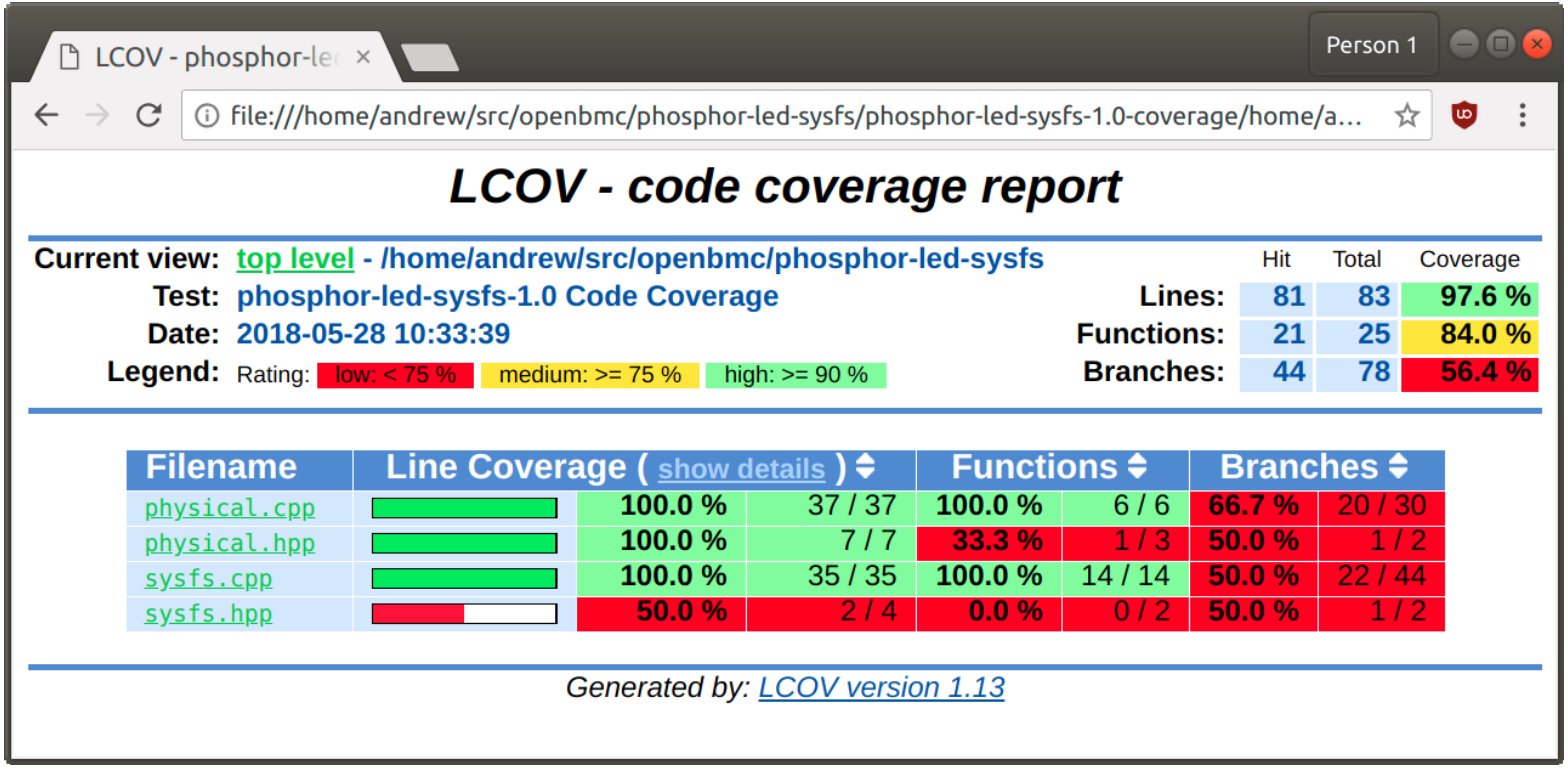
Enforce behaviour with Sensing

https://gerrit.openbmc-project.xyz/#/c/10836/

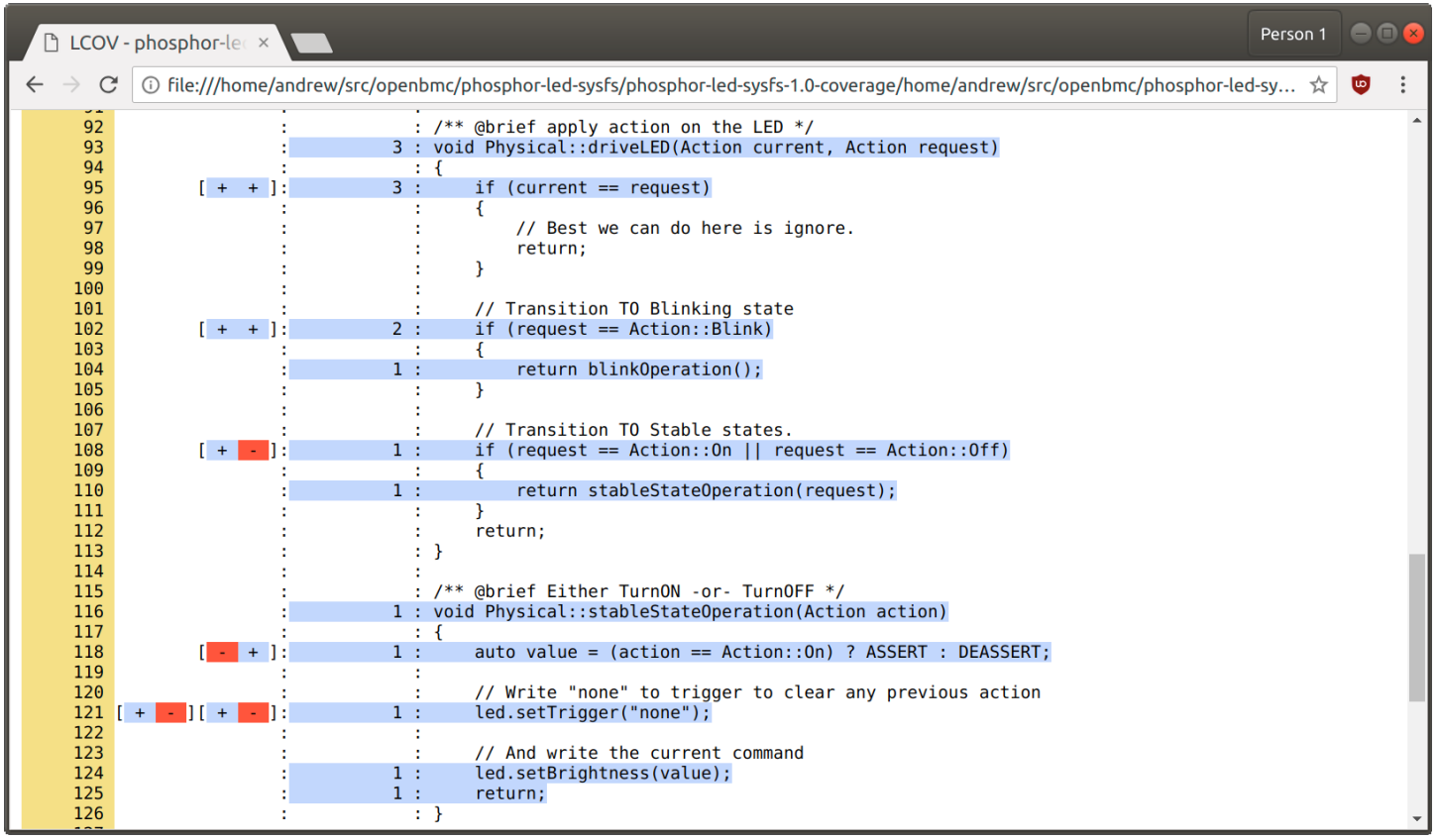
```
diff --git a/test/physical.cpp b/test/physical.cpp
index fade5c8c3bd9..bbb51438c722 100644
--- a/test/physical.cpp
+++ b/test/physical.cpp
@@ -110,7 +110,10 @@ TEST(Physical, blink)
 {
     sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
     NiceMock<MockLed> led;
-    ON_CALL(led, getTrigger()).WillByDefault(Return("none"));
+    EXPECT_CALL(led, getTrigger()).WillOnce(Return("none"));
+    EXPECT_CALL(led, setTrigger("timer"));
+    EXPECT_CALL(led, setDelayOn(500));
+    EXPECT_CALL(led, setDelayOff(500));
     phosphor::led::Physical phy(bus, LED_OBJ, led);
     phy.state(Action::Blink);
 }
```

Branch Coverage

make **CODE_COVERAGE_BRANCH_COVERAGE=1** check-code-coverage



Branch Coverage



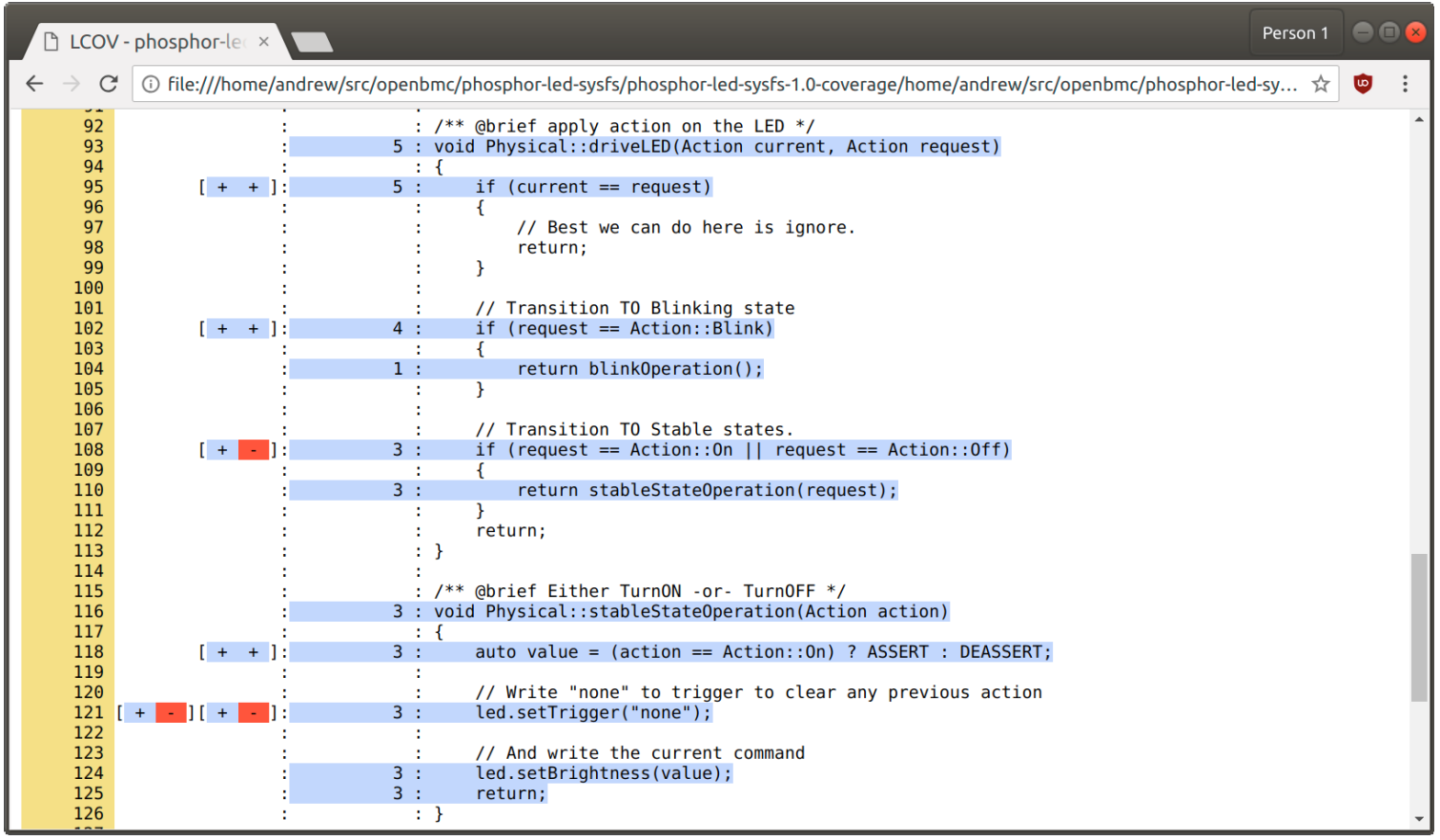
Covering un-hit branches

https://gerrit.openbmc-project.xyz/#/c/10836/

```
diff --git a/test/physical.cpp b/test/physical.cpp
index 156a2e5e87d1..fade5c8c3bd9 100644
--- a/test/physical.cpp
+++ b/test/physical.cpp
@@ -64,6 +64,7 @@ class MockLed : public phosphor::led::SysfsLed
     MOCK_METHOD1(setDelayOff, void(unsigned long ms));
 };

+using ::testing::InSequence;
 using ::testing::NiceMock;
 using ::testing::Return;
 using ::testing::Throw;
@@ -123,3 +124,20 @@ TEST(Physical, ctor_none_trigger_asserted_brightness)
     EXPECT_CALL(led, getBrightness()).WillRepeatedly(Return(val));
     phosphor::led::Physical phy(bus, LED_OBJ, led);
 }
+
+TEST(Physical, on_to_off)
+{
+    InSequence s;
+
+    sdbusplus::bus::bus bus = sdbusplus::bus::new_default();
+    NiceMock<MockLed> led;
+    EXPECT_CALL(led, getTrigger()).Times(1).WillOnce(Return("none"));
+    unsigned long deasserted = phosphor::led::DEASSERT;
+    EXPECT_CALL(led, getBrightness()).WillOnce(Return(deasserted));
+    unsigned long asserted = phosphor::led::ASSERT;
+    EXPECT_CALL(led, setBrightness(asserted));
+    EXPECT_CALL(led, setBrightness(deasserted));
+    phosphor::led::Physical phy(bus, LED_OBJ, led);
+    phy.state(Action::On);
+    phy.state(Action::Off);
+}
```

The eighth result




```
                  :           :  /** @brief apply action on the LED */
                  :         5 :  void Physical::driveLED(Action current, Action request)
                  :           :  {
        [ +  + ]  :         5 :      if (current == request)
                  :           :      {
                  :           :          // Best we can do here is ignore.
                  :           :          return;
                  :           :      }
                  :           :
                  :           :      // Transition TO Blinking state
        [ +  + ]  :         4 :      if (request == Action::Blink)
                  :           :      {
                  :         1 :          return blinkOperation();
                  :           :      }
                  :           :
                  :           :      // Transition TO Stable states.
        [ +  - ]  :         3 :      if (request == Action::On || request == Action::Off)
                  :           :      {
                  :         3 :          return stableStateOperation(request);
                  :           :      }
                  :           :      return;
                  :           :  }
                  :           :
                  :           :  /** @brief Either TurnON -or- TurnOFF */
                  :         3 :  void Physical::stableStateOperation(Action action)
                  :           :  {
        [ +  + ]  :         3 :      auto value = (action == Action::On) ? ASSERT : DEASSERT;
                  :           :
                  :           :      // Write "none" to trigger to clear any previous action
[ +  - ][ +  - ]  :         3 :      led.setTrigger("none");
                  :           :
                  :           :      // And write the current command
                  :         3 :      led.setBrightness(value);
                  :         3 :      return;
                  :           :  }
```

Now that tests are in place...

- physical: Conform to LED class kernel ABI
- physical: Avoid unreachable statement in driveLED()
- physical: Cleanup unnecessary variables
- physical: Rework commentary for brevity
- physical: 'frequency' is really periodicity

Ratcheting up the pressure

https://gerrit.openbmc-project.xyz/#/c/10828/

```
diff --git a/bootstrap.sh b/bootstrap.sh
index 50b75b7ee911..11c8ae9f96c0 100755
--- a/bootstrap.sh
+++ b/bootstrap.sh
@@ -1,10 +1,20 @@
 #!/bin/sh

+set -eu
+
 AUTOCONF_FILES="Makefile.in aclocal.m4 ar-lib autom4te.cache compile \
         config.guess config.h.in config.sub configure depcomp install-sh \
         ltmain.sh missing *libtool test-driver"

-case $1 in
+BOOTSTRAP_MODE=""
+
+if [ $# -gt 0 ];
+then
+    BOOTSTRAP_MODE="${1}"
+    shift 1
+fi
+
+case ${BOOTSTRAP_MODE} in
     clean)
         test -f Makefile && make maintainer-clean
         for file in ${AUTOCONF_FILES}; do
@@ -15,4 +25,17 @@ case $1 in
 esac

 autoreconf -i
-echo 'Run "./configure ${CONFIGURE_FLAGS} && make"'
+
+case ${BOOTSTRAP_MODE} in
+    dev)
+        FLAGS="-fsanitize=address -fsanitize=leak -fsanitize=undefined -Wall -Werror"
+        ./configure CFLAGS="${FLAGS}"  CXXFLAGS="${FLAGS}"  --enable-code-coverage "$@"
+        ;;
+    *)
+        echo 'Run "./configure ${CONFIGURE_FLAGS} && make"'
+        ;;
+esac
```

Code Sanitizers

- -fsanitize=address
  - Immediate abort and report on bad memory accesses
- -fsanitize=leak
  - Memory leak report on application termination
- -fsanitize=undefined
  - Immediate abort and report on undefined behaviour

Wrapping up...

- Automated testing is automated code review
  - Accelerates future development through characterisation
  - Good coverage drastically reduces risk of (invasive) changes
- You need to instrument your tests
  - Use code coverage metrics to guide your testing
- Rework abstractions to increase coverage
  - Dependency injection allows for mocks and thus sensing
- Use branch coverage analysis to find missed paths
- Use sanitizers so your tests fail fast, hard and informatively
- Use this presentation to adopt tests in your code!

Recommended Reading

- Working Effectively with Legacy Code
- Design Patterns: Elements of Reusable Object-Oriented Software

Please attribute IBM Corp when using or adapting this work