GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	src/	**Lines:**	0	79	0.0 %
Date:	2022-03-19 22:39:16	**Branches:**	0	0	- %
Legend:	low: < 75.0 % medium: >= 75.0 % high: >= 90.0 %				

File	Lines			Branches	
adg2188.cpp		0.0 %	0 / 79	- %	0 / 0

Generated by: GCOVR (Version 4.2)

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	src/	**Lines:**	0	79	0.0 %
Date:	2022-03-19 22:39:16	**Branches:**	0	0	- %
Legend:	low: < 75.0 % medium: >= 75.0 % high: >= 90.0 %				

File	Lines			Branches	
adg2188.cpp		0.0 %	0 / 79	- %	0 / 0

Generated by: GCOVR (Version 4.2)

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	src/				
File:	src/adg2188.cpp	Lines:	0	79	0.0 %
Date:	2022-03-19 22:39:16	Branches:	0	0	- %

```
Line Branch Exec  Source
   1            //  MIT License
   2            //
   3            //  Copyright (c) 2022 Chris Sutton
   4            //
   5            //  Permission is hereby granted, free of charge, to any person obtaining a copy
   6            //  of this software and associated documentation files (the "Software"), to deal
   7            //  in the Software without restriction, including without limitation the rights
   8            //  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
   9            //  copies of the Software, and to permit persons to whom the Software is
  10            //  furnished to do so, subject to the following conditions:
  11            //
  12            //  The above copyright notice and this permission notice shall be included in all
  13            //  copies or substantial portions of the Software.
  14            //
  15            //  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
  16            //  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
  17            //  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
  18            //  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
  19            //  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
  20            //  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
  21            //  SOFTWARE.
  22
  23            #include <adg2188.hpp>
  24
  25
  26            namespace adg2188
  27            {
  28
  29        #####   Driver::Driver(I2C_TypeDef *i2c_handle)
  30            {
  31                // m_i2c_handle = std::unique_ptr<I2C_TypeDef>(i2c_handle);
  32        #####       m_i2c_handle = i2c_handle;
  33        #####       probe_i2c();
  34        #####       clear_all();
  35
  36                    // // close/open asynchronously, i.e. one at a time
  37                    // read_xline_switch_values(XLineRead::X2);
  38                    // write_switch(Throw::close, Pole::x2_to_y0, Latch::set);
  39                    // read_xline_switch_values(XLineRead::X2);
  40                    // read_xline_switch_values(XLineRead::X1);
  41                    // write_switch(Throw::close, Pole::x1_to_y0, Latch::set);
  42                    // read_xline_switch_values(XLineRead::X1);
  43                    // read_xline_switch_values(XLineRead::X2);
  44                    // write_switch(Throw::open, Pole::x2_to_y0, Latch::set);
  45                    // read_xline_switch_values(XLineRead::X2);
  46                    // read_xline_switch_values(XLineRead::X1);
  47                    // write_switch(Throw::open, Pole::x1_to_y0, Latch::set);
  48                    // read_xline_switch_values(XLineRead::X1);
  49
  50                    // // close/open synchronously, i.e. at the same time
  51                    // read_xline_switch_values(XLineRead::X1);
  52                    // read_xline_switch_values(XLineRead::X2);
  53                    // write_switch(Throw::close, Pole::x2_to_y0, Latch::reset);
  54                    // write_switch(Throw::close, Pole::x1_to_y0, Latch::set);
  55
  56                    // write_switch(Throw::open, Pole::x2_to_y0, Latch::reset);
  57                    // write_switch(Throw::open, Pole::x1_to_y0, Latch::set);
  58                    // read_xline_switch_values(XLineRead::X1);
  59                    // read_xline_switch_values(XLineRead::X2);
  60
  61            }
  62
  63        #####   bool Driver::clear_all()
  64            {
  65        #####       write_switch(Throw::open, Pole::x0_to_y0, Latch::set);
  66        #####       write_switch(Throw::open, Pole::x1_to_y0, Latch::set);
  67        #####       write_switch(Throw::open, Pole::x2_to_y0, Latch::set);
  68        #####       write_switch(Throw::open, Pole::x3_to_y0, Latch::set);
  69        #####       write_switch(Throw::open, Pole::x4_to_y0, Latch::set);
  70        #####       write_switch(Throw::open, Pole::x5_to_y0, Latch::set);
  71        #####       write_switch(Throw::open, Pole::x6_to_y0, Latch::set);
  72        #####       write_switch(Throw::open, Pole::x7_to_y0, Latch::set);
  73
  74        #####       write_switch(Throw::open, Pole::x0_to_y1, Latch::set);
  75        #####       write_switch(Throw::open, Pole::x1_to_y1, Latch::set);
  76        #####       write_switch(Throw::open, Pole::x2_to_y1, Latch::set);
  77        #####       write_switch(Throw::open, Pole::x3_to_y1, Latch::set);
  78        #####       write_switch(Throw::open, Pole::x4_to_y1, Latch::set);
  79        #####       write_switch(Throw::open, Pole::x5_to_y1, Latch::set);
  80        #####       write_switch(Throw::open, Pole::x6_to_y1, Latch::set);
  81        #####       write_switch(Throw::open, Pole::x7_to_y1, Latch::set);
  82
  83        #####       write_switch(Throw::open, Pole::x0_to_y2, Latch::set);
  84        #####       write_switch(Throw::open, Pole::x1_to_y2, Latch::set);
  85        #####       write_switch(Throw::open, Pole::x2_to_y2, Latch::set);
  86        #####       write_switch(Throw::open, Pole::x3_to_y2, Latch::set);
  87        #####       write_switch(Throw::open, Pole::x4_to_y2, Latch::set);
  88        #####       write_switch(Throw::open, Pole::x5_to_y2, Latch::set);
  89        #####       write_switch(Throw::open, Pole::x6_to_y2, Latch::set);
  90        #####       write_switch(Throw::open, Pole::x7_to_y2, Latch::set);
  91
  92        #####       write_switch(Throw::open, Pole::x0_to_y3, Latch::set);
  93        #####       write_switch(Throw::open, Pole::x1_to_y3, Latch::set);
  94        #####       write_switch(Throw::open, Pole::x2_to_y3, Latch::set);
  95        #####       write_switch(Throw::open, Pole::x3_to_y3, Latch::set);
  96        #####       write_switch(Throw::open, Pole::x4_to_y3, Latch::set);
  97        #####       write_switch(Throw::open, Pole::x5_to_y3, Latch::set);
```

```
    write_switch(Throw::open, Pole::x6_to_y3, Latch::set);
    write_switch(Throw::open, Pole::x7_to_y3, Latch::set);

    write_switch(Throw::open, Pole::x0_to_y4, Latch::set);
    write_switch(Throw::open, Pole::x1_to_y4, Latch::set);
    write_switch(Throw::open, Pole::x2_to_y4, Latch::set);
    write_switch(Throw::open, Pole::x3_to_y4, Latch::set);
    write_switch(Throw::open, Pole::x4_to_y4, Latch::set);
    write_switch(Throw::open, Pole::x5_to_y4, Latch::set);
    write_switch(Throw::open, Pole::x6_to_y4, Latch::set);
    write_switch(Throw::open, Pole::x7_to_y4, Latch::set);

    write_switch(Throw::open, Pole::x0_to_y5, Latch::set);
    write_switch(Throw::open, Pole::x1_to_y5, Latch::set);
    write_switch(Throw::open, Pole::x2_to_y5, Latch::set);
    write_switch(Throw::open, Pole::x3_to_y5, Latch::set);
    write_switch(Throw::open, Pole::x4_to_y5, Latch::set);
    write_switch(Throw::open, Pole::x5_to_y5, Latch::set);
    write_switch(Throw::open, Pole::x6_to_y5, Latch::set);
    write_switch(Throw::open, Pole::x7_to_y5, Latch::set);

    write_switch(Throw::open, Pole::x0_to_y6, Latch::set);
    write_switch(Throw::open, Pole::x1_to_y6, Latch::set);
    write_switch(Throw::open, Pole::x2_to_y6, Latch::set);
    write_switch(Throw::open, Pole::x3_to_y6, Latch::set);
    write_switch(Throw::open, Pole::x4_to_y6, Latch::set);
    write_switch(Throw::open, Pole::x5_to_y6, Latch::set);
    write_switch(Throw::open, Pole::x6_to_y6, Latch::set);
    write_switch(Throw::open, Pole::x7_to_y6, Latch::set);

    write_switch(Throw::open, Pole::x0_to_y7, Latch::set);
    write_switch(Throw::open, Pole::x1_to_y7, Latch::set);
    write_switch(Throw::open, Pole::x2_to_y7, Latch::set);
    write_switch(Throw::open, Pole::x3_to_y7, Latch::set);
    write_switch(Throw::open, Pole::x4_to_y7, Latch::set);
    write_switch(Throw::open, Pole::x5_to_y7, Latch::set);
    write_switch(Throw::open, Pole::x6_to_y7, Latch::set);
    write_switch(Throw::open, Pole::x7_to_y7, Latch::set);
    return true;
}

bool Driver::probe_i2c()
{
 bool success {true};

    // check ADG2188 is listening on 0xE0.
    #ifndef X86_UNIT_TESTING_ONLY
 if (stm32::i2c::send_addr(m_i2c_handle, i2c_addr, stm32::i2c::MsgType::PROBE) == stm32::i2c::Status::NACK)
    {
        success = false;
    }
    #endif
    return success;

}

// See page 20 of https://www.analog.com/media/en/technical-documentation/data-sheets/adg2188.pdf
bool Driver::write_switch(const Throw &sw_throw [[maybe_unused]], const Pole &sw_pole [[maybe_unused]], const Latch &sw_latch [[maybe_unused]])
{
    bool success {true};

#if not defined(X86_UNIT_TESTING_ONLY)
    // write this number of bytes: The data byte(s) AND the address byte
    stm32::i2c::set_numbytes(m_i2c_handle, 2);

    // check ADG2188 is listening on 0xE0 + 1.
 if (stm32::i2c::send_addr(m_i2c_handle, i2c_addr, stm32::i2c::MsgType::WRITE) == stm32::i2c::Status::NACK)
    {
        success = false;
    }

    // switch config byte
    uint8_t switch_configuration = (static_cast<uint8_t>(sw_throw) | static_cast<uint8_t>(sw_pole));
    stm32::i2c::send_byte(m_i2c_handle, switch_configuration);
    // latch byte
    if (sw_latch == Latch::set)
    {
        stm32::i2c::send_byte(m_i2c_handle, 0x01);
    }
    else
    {
        stm32::i2c::send_byte(m_i2c_handle, 0x00);
    }

    // Generate the stop condition
 stm32::i2c::generate_stop_condition(m_i2c_handle);
#endif

    return success;
}

// See page 22 of https://www.analog.com/media/en/technical-documentation/data-sheets/adg2188.pdf
bool Driver::read_xline_switch_values(XLineRead line [[maybe_unused]])
{
    bool success {true};

#if not defined(X86_UNIT_TESTING_ONLY)
    // write this number of bytes: The data byte(s) AND the address byte
 stm32::i2c::set_numbytes(m_i2c_handle, 2);

    // check ADG2188 is listening on 0xE0 + 1.
 if (stm32::i2c::send_addr(m_i2c_handle, i2c_addr, stm32::i2c::MsgType::WRITE) == stm32::i2c::Status::NACK)
    {
        success = false;
    }
    // request the xline we want to read (second byte is don't care, so just repeat it)
    stm32::i2c::send_byte(m_i2c_handle, static_cast<uint8_t>(line));
```

```
205                 stm32::i2c::send_byte(m_i2c_handle, static_cast<uint8_t>(line));
206
207                 // Generate the stop condition
208              stm32::i2c::generate_stop_condition(m_i2c_handle);
209
210                 // check ADG2188 is listening on 0xE0 + 0.
211              if (stm32::i2c::send_addr(m_i2c_handle, i2c_addr, stm32::i2c::MsgType::READ) == stm32::i2c::Status::NACK)
212                 {
213                     success = false;
214                 }
215
216                 // receive the first byte and send back ACk to slave
217                 uint8_t rx_byte1 {0};
218                 stm32::i2c::receive_byte(m_i2c_handle, rx_byte1);
219                 stm32::i2c::send_ack(m_i2c_handle);
220
221                 // receive the second byte, send NACK and STOP to slave
222                 uint8_t rx_byte2 {0};
223                 stm32::i2c::receive_byte(m_i2c_handle, rx_byte2);
224                 stm32::i2c::send_nack(m_i2c_handle);
225
226              stm32::i2c::generate_stop_condition(m_i2c_handle);
227             #endif
228                 return success;
229             }
230
231             } // namespace adg2188
```