GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	src/	**Lines:**	0	63	0.0 %
Date:	2022-03-20 00:42:26	**Branches:**	0	74	0.0 %
Legend:	low: < 75.0 % medium: >= 75.0 % high: >= 90.0 %				

File	Lines			Branches	
tlc5955.cpp		0.0 %	0 / 63	0.0 %	0 / 74

Generated by: GCOVR (Version 4.2)

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	src/	**Lines:**	0	63	0.0 %
Date:	2022-03-20 00:42:26	**Branches:**	0	74	0.0 %
Legend:	low: < 75.0 % medium: >= 75.0 % high: >= 90.0 %				

File	Lines			Branches	
tlc5955.cpp		0.0 %	0 / 63	0.0 %	0 / 74

Generated by: GCOVR (Version 4.2)

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	src/				
File:	src/tlc5955.cpp	Lines:	0	63	0.0 %
Date:	2022-03-20 00:42:26	Branches:	0	74	0.0 %

```
Line Branch Exec  Source
   1                
   2                #include "tlc5955.hpp"
   3                
   4                #include <cmath>
   5                #include <cstring>
   6                #include <cassert>
   7                
   8                #if defined(X86_UNIT_TESTING_ONLY)
   9                    #if defined(USE_RTT)
  10                        #include <SEGGER_RTT.h>
  11                    #endif
  12                #else
  13                    #include <spi_utils.hpp>
  14                #endif
  15               
  16               namespace tlc5955
  17               {
  18               
  19   ######      Driver::Driver(const DriverSerialInterface &serial_interface) : m_serial_interface(serial_interface)
  20               {
  21                   #if not defined(X86_UNIT_TESTING_ONLY)
  22                   #pragma GCC diagnostic push
  23                   #pragma GCC diagnostic ignored "-Wvolatile"
  24               
  25                       // Setup GPIO clock. Used to send first bit
  26                       __IO uint32_t tmpreg;
  27                       RCC->IOPENR = RCC->IOPENR | m_serial_interface.get_rcc_gpio_clk();
  28                       tmpreg = (RCC->IOPENR & m_serial_interface.get_rcc_gpio_clk());
  29                       (void)tmpreg;
  30               
  31                       // Setup SPI clock. Used to send subsequent 96 bytes over SPI
  32                       SET_BIT(RCC->APBENR1, m_serial_interface.get_rcc_spi_clk());
  33               
  34                   #pragma GCC diagnostic pop  // ignored "-Wvolatile"
  35                   #endif // not X86_UNIT_TESTING_ONLY
  36               }
  37               
  38               // @brief class to implement TLC5955 LED Driver IC
  39               // Refer to datasheet - https://www.ti.com/lit/ds/symlink/tlc5955.pdf
  40   ######      void Driver::reset()
  41               {
  42   ######          m_common_bit_register.reset();
  43   ######          m_common_byte_register.fill(0);
  44               }
  45               
  46   ######      void Driver::send_first_bit(DataLatchType latch_type [[maybe_unused]])
  47               {
  48               #if not defined(X86_UNIT_TESTING_ONLY)
  49                   stm32::spi::enable_spi(m_serial_interface.get_spi_handle(), false);
  50               
  51                   // set PB7/PB8 as GPIO outputs
  52                   gpio_init();
  53               
  54                   // make sure LAT pin is low otherwise first latch may be skipped (and TLC5955 will initialise intermittently)
  55                   LL_GPIO_ResetOutputPin(m_serial_interface.get_lat_port(), m_serial_interface.get_lat_pin());
  56               
  57                   // "Control Data Latch" - Start SPI transacation by clocking in one high bit
  58                   if (latch_type == DataLatchType::control)
  59                   {
  60                       // reset both SCK and MOSI
  61                       LL_GPIO_ResetOutputPin(m_serial_interface.get_mosi_port(), m_serial_interface.get_mosi_pin());
  62                       LL_GPIO_ResetOutputPin(m_serial_interface.get_mosi_port(), m_serial_interface.get_sck_pin());
  63               
  64                       // MOSI data clocked on high(1) rising edge of SCK
  65                       LL_GPIO_SetOutputPin(m_serial_interface.get_mosi_port(), m_serial_interface.get_mosi_pin());
  66                       LL_GPIO_SetOutputPin(m_serial_interface.get_mosi_port(), m_serial_interface.get_sck_pin());
  67               
  68               
  69                       // reset both SCK and MOSI
  70                       LL_GPIO_ResetOutputPin(m_serial_interface.get_mosi_port(), m_serial_interface.get_mosi_pin());
  71                       LL_GPIO_ResetOutputPin(m_serial_interface.get_mosi_port(), m_serial_interface.get_sck_pin());
  72               
  73               
  74                   }
  75                   // "GS Data Latch" - Start SPI transacation by clocking in one low bit
  76                   else
  77                   {
  78                       // reset both SCK and MOSI
  79                       LL_GPIO_ResetOutputPin(m_serial_interface.get_mosi_port(), m_serial_interface.get_sck_pin());
  80                       LL_GPIO_ResetOutputPin(m_serial_interface.get_mosi_port(), m_serial_interface.get_mosi_pin());
  81               
  82                       // MOSI data clocked low(0) on rising edge of SCK
  83                       LL_GPIO_SetOutputPin(m_serial_interface.get_mosi_port(), m_serial_interface.get_sck_pin());
  84                       LL_GPIO_ResetOutputPin(m_serial_interface.get_mosi_port(), m_serial_interface.get_mosi_pin());
  85               
  86                       // reset both SCK and MOSI
  87                       LL_GPIO_ResetOutputPin(m_serial_interface.get_mosi_port(), m_serial_interface.get_sck_pin());
  88                       LL_GPIO_ResetOutputPin(m_serial_interface.get_mosi_port(), m_serial_interface.get_mosi_pin());
  89                   }
  90               
  91                   // set PB7/PB8 to SPI
  92                   spi2_init();
  93                   stm32::spi::enable_spi(m_serial_interface.get_spi_handle());
  94               
  95               #endif
  96               }
  97               
  98   ######      void Driver::set_padding_bits()
  99               {
 100   ######          noarch::bit_manip::insert_bitset_at_offset(m_common_bit_register, m_padding, m_padding_offset);
 101               }
 102              
 103   ######      void Driver::set_ctrl_cmd()
 104              {
 105   ######          noarch::bit_manip::insert_bitset_at_offset(m_common_bit_register, m_ctrl_cmd, m_ctrl_cmd_offset);
 106              }
 107              
 108   ######     void Driver::set_function_cmd(DisplayFunction dsprpt, TimingFunction tmgrst, RefreshFunction rfresh, PwmFunction espwm, ShortDetectFunction lsdvlt)
 109              {
 110   ######         std::bitset<m_func_cmd_size> function_cmd {};
 111   ######         (dsprpt == DisplayFunction::display_repeat_on)          ? function_cmd.set(4, true) : function_cmd.set(4, false);
 112   ######         (tmgrst == TimingFunction::timing_reset_on)             ? function_cmd.set(3, true) : function_cmd.set(3, false);
 113   ######         (rfresh == RefreshFunction::auto_refresh_on)            ? function_cmd.set(2, true) : function_cmd.set(2, false);
```

```
    (espwm == PwmFunction::enhanced_pwm)                    ? function_cmd.set(1, true) : function_cmd.set(1, false);
    (lsdvlt == ShortDetectFunction::threshold_90_percent)   ? function_cmd.set(0, true) : function_cmd.set(0, false);
    noarch::bit_manip::insert_bitset_at_offset(m_common_bit_register, function_cmd, m_func_cmd_offset);
}

void Driver::set_global_brightness_cmd(const uint8_t blue, const uint8_t green, const uint8_t red)
{
    const std::bitset<m_bc_data_size> blue_cmd {blue};
    const std::bitset<m_bc_data_size> green_cmd {green};
    const std::bitset<m_bc_data_size> red_cmd {red};

    noarch::bit_manip::insert_bitset_at_offset(m_common_bit_register, blue_cmd, m_bc_data_offset);
    noarch::bit_manip::insert_bitset_at_offset(m_common_bit_register, green_cmd, m_bc_data_offset + m_bc_data_size);
    noarch::bit_manip::insert_bitset_at_offset(m_common_bit_register, red_cmd, m_bc_data_offset + m_bc_data_size * 2);

}

void Driver::set_max_current_cmd(const uint8_t blue, const uint8_t green, const uint8_t red)
{
    const std::bitset<m_mc_data_size> blue_cmd {blue};
    const std::bitset<m_mc_data_size> green_cmd {green};
    const std::bitset<m_mc_data_size> red_cmd {red};

    noarch::bit_manip::insert_bitset_at_offset(m_common_bit_register, blue_cmd, m_mc_data_offset);
    noarch::bit_manip::insert_bitset_at_offset(m_common_bit_register, green_cmd, m_mc_data_offset + m_mc_data_size);
    noarch::bit_manip::insert_bitset_at_offset(m_common_bit_register, red_cmd, m_mc_data_offset + m_mc_data_size * 2);
}

void Driver::set_dot_correction_cmd_all(uint8_t pwm)
{
    const std::bitset<m_dc_data_size> dc_pwm_cmd {pwm};
    for (uint8_t dc_idx = 0; dc_idx < 48; dc_idx++)

 {
  noarch::bit_manip::insert_bitset_at_offset(m_common_bit_register, dc_pwm_cmd, m_dc_data_offset + m_dc_data_size * dc_idx);
 }
}

void Driver::set_greyscale_cmd_rgb(uint16_t blue_pwm, uint16_t green_pwm, uint16_t red_pwm)
{
    const std::bitset<m_gs_data_size> blue_gs_pwm_cmd {blue_pwm};
    const std::bitset<m_gs_data_size> green_gs_pwm_cmd {green_pwm};
    const std::bitset<m_gs_data_size> red_gs_pwm_cmd {red_pwm};
    for (uint16_t gs_idx = 0; gs_idx < m_num_leds_per_chip; gs_idx++)
    {
     noarch::bit_manip::insert_bitset_at_offset(m_common_bit_register, blue_gs_pwm_cmd, m_gs_data_offset   + (m_gs_data_size * gs_idx * m_num_colour_chan));
        noarch::bit_manip::insert_bitset_at_offset(m_common_bit_register, green_gs_pwm_cmd, m_gs_data_offset   + (m_gs_data_size * gs_idx * m_num_colour_chan) + m_gs_d
        noarch::bit_manip::insert_bitset_at_offset(m_common_bit_register, red_gs_pwm_cmd, m_gs_data_offset   + (m_gs_data_size * gs_idx * m_num_colour_chan) + (m_gs_da
    }
}

void Driver::set_greyscale_cmd_white(uint16_t pwm)
{
    const std::bitset<m_gs_data_size> gs_pwm_cmd {pwm};
    for (uint16_t gs_idx = 0; gs_idx < 48; gs_idx++)
    {
     noarch::bit_manip::insert_bitset_at_offset(m_common_bit_register, gs_pwm_cmd, m_gs_data_offset + m_gs_data_size * gs_idx);
    }
}

bool Driver::set_greyscale_cmd_rgb_at_position(uint16_t led_idx, uint16_t red_pwm, uint16_t green_pwm, uint16_t blue_pwm)
{
    // return if we overshot our max number of LEDs
    if (! (led_idx < tlc5955::Driver::m_num_leds_per_chip))
    {
        return false;
    }

    const std::bitset<m_gs_data_size> blue_gs_pwm_cmd {blue_pwm};
    const std::bitset<m_gs_data_size> green_gs_pwm_cmd {green_pwm};
    const std::bitset<m_gs_data_size> red_gs_pwm_cmd {red_pwm};

    noarch::bit_manip::insert_bitset_at_offset(m_common_bit_register, blue_gs_pwm_cmd, m_gs_data_offset   + (m_gs_data_size * led_idx * m_num_colour_chan));
    noarch::bit_manip::insert_bitset_at_offset(m_common_bit_register, green_gs_pwm_cmd, m_gs_data_offset   + (m_gs_data_size * led_idx * m_num_colour_chan) + m_gs_data
    noarch::bit_manip::insert_bitset_at_offset(m_common_bit_register, red_gs_pwm_cmd, m_gs_data_offset   + (m_gs_data_size * led_idx * m_num_colour_chan) + (m_gs_data_
    return true;
}

bool Driver::send_spi_bytes(LatchPinOption latch_option [[maybe_unused]])
{
#if not defined(X86_UNIT_TESTING_ONLY)
    // send the bytes
    for (auto &byte: m_common_byte_register)
    {
        // send the byte of data
        stm32::spi::send_byte(m_serial_interface.get_spi_handle(), byte);
    }

    // tell each daisy-chained driver chip to latch all data from its common register
    if (latch_option == LatchPinOption::latch_after_send)
    {
        LL_GPIO_SetOutputPin(m_serial_interface.get_lat_port(), m_serial_interface.get_lat_pin());
        LL_GPIO_ResetOutputPin(m_serial_interface.get_lat_port(), m_serial_interface.get_lat_pin());
    }
#endif
    return true;

}

void Driver::process_register()
{
    // noarch::bit_manip::print_bits(m_common_bit_register);
    noarch::bit_manip::bitset_to_bytearray(m_common_byte_register, m_common_bit_register);
    // noarch::byte_manip::print_bytes(m_common_byte_register);
}

void Driver::gpio_init(void)
{
    #if not defined(X86_UNIT_TESTING_ONLY)
        LL_GPIO_InitTypeDef GPIO_InitStruct = {0,0,0,0,0,0};



        // TLC5955_SPI2_MOSI
        //LL_GPIO_SetOutputPin(m_serial_interface.get_mosi_port(), m_serial_interface.get_mosi_pin());
        GPIO_InitStruct.Pin = m_serial_interface.get_mosi_pin();
        GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
        GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_VERY_HIGH;
        GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
        GPIO_InitStruct.Pull = LL_GPIO_PULL_DOWN;
        LL_GPIO_Init(m_serial_interface.get_mosi_port(), &GPIO_InitStruct);

        // TLC5955_SPI2_SCK
```

```
235             // LL_GPIO_ResetOutputPin(m_serial_interface.get_sck_port(), m_serial_interface.get_sck_pin());
236             GPIO_InitStruct.Pin = m_serial_interface.get_sck_pin();
237             GPIO_InitStruct.Mode = LL_GPIO_MODE_OUTPUT;
238             GPIO_InitStruct.Speed = LL_GPIO_SPEED_FREQ_VERY_HIGH;
239             GPIO_InitStruct.OutputType = LL_GPIO_OUTPUT_PUSHPULL;
240             GPIO_InitStruct.Pull = LL_GPIO_PULL_UP;
241             LL_GPIO_Init(m_serial_interface.get_sck_port(), &GPIO_InitStruct);
242         #endif // not X86_UNIT_TESTING_ONLY
243     }
244     void Driver::spi2_init(void)
245     {
246         #if not defined(X86_UNIT_TESTING_ONLY)
247         #pragma GCC diagnostic push
248         #pragma GCC diagnostic ignored "-Wvolatile"
249             // Enable GPIO (SPI_MOSI)
250             LL_GPIO_SetPinSpeed(m_serial_interface.get_mosi_port(), m_serial_interface.get_mosi_pin(), LL_GPIO_SPEED_FREQ_VERY_HIGH);
251             LL_GPIO_SetPinOutputType(m_serial_interface.get_mosi_port(), m_serial_interface.get_mosi_pin(), LL_GPIO_OUTPUT_PUSHPULL);
252             LL_GPIO_SetPinPull(m_serial_interface.get_mosi_port(), m_serial_interface.get_mosi_pin(), LL_GPIO_PULL_DOWN);
253             LL_GPIO_SetAFPin_0_7(m_serial_interface.get_mosi_port(), m_serial_interface.get_mosi_pin(), LL_GPIO_AF_1);
254             LL_GPIO_SetPinMode(m_serial_interface.get_mosi_port(), m_serial_interface.get_mosi_pin(), LL_GPIO_MODE_ALTERNATE);
255
256             // Enable GPIO (SPI_SCK)
257             LL_GPIO_SetPinSpeed(m_serial_interface.get_sck_port(), m_serial_interface.get_sck_pin(), LL_GPIO_SPEED_FREQ_VERY_HIGH);
258             LL_GPIO_SetPinOutputType(m_serial_interface.get_sck_port(), m_serial_interface.get_sck_pin(), LL_GPIO_OUTPUT_PUSHPULL);
259             LL_GPIO_SetPinPull(m_serial_interface.get_sck_port(), m_serial_interface.get_sck_pin(), LL_GPIO_PULL_DOWN);
260             LL_GPIO_SetAFPin_8_15(m_serial_interface.get_sck_port(), m_serial_interface.get_sck_pin(), LL_GPIO_AF_1);
261             LL_GPIO_SetPinMode(m_serial_interface.get_sck_port(), m_serial_interface.get_sck_pin(), LL_GPIO_MODE_ALTERNATE);
262
263             m_serial_interface.get_spi_handle()->CR1 = 0;
264             m_serial_interface.get_spi_handle()->CR1 |=
265                 ((SPI_CR1_BIDIMODE | SPI_CR1_BIDIOE) | (SPI_CR1_MSTR | SPI_CR1_SSI) | SPI_CR1_SSM | SPI_CR1_BR_1);
266
267             CLEAR_BIT(m_serial_interface.get_spi_handle()->CR2, SPI_CR2_NSSP);
268
269             // Enable the PWM OC channel
270             m_serial_interface.get_gsclk_handle()->CCER = m_serial_interface.get_gsclk_handle()->CCER | m_serial_interface.get_gsclk_tim_ch();
271             // required to enable output on some timers. e.g. TIM16
272             m_serial_interface.get_gsclk_handle()->BDTR = m_serial_interface.get_gsclk_handle()->BDTR | TIM_BDTR_MOE;
273             // Enable the timer
274             m_serial_interface.get_gsclk_handle()->CR1 = m_serial_interface.get_gsclk_handle()->CR1 | TIM_CR1_CEN;
275
276
277         #pragma GCC diagnostic pop  // ignored "-Wvolatile"
278         #endif // not X86_UNIT_TESTING_ONLY
279
280     }
281
282
283 } // namespace tlc5955
```