GCC Code Coverage Report

Directory:	./		**Exec**	**Total**	**Coverage**
Date:	2021-12-05 16:22:01	**Lines:**	311	354	87.9 %
Legend:	low: < 75.0 % medium: >= 75.0 % high: >= 90.0 %	**Branches:**	234	501	46.7 %

File	Lines			Branches	
cpp_ssd1306/inc/font.hpp		100.0 %	8 / 8	100.0 %	2 / 2
cpp_ssd1306/inc/ssd1306.hpp		86.0 %	43 / 50	14.5 %	9 / 62
cpp_ssd1306/src/ssd1306.cpp		100.0 %	65 / 65	57.9 %	44 / 76
cpp_tlc5955/inc/byte_position.hpp		100.0 %	23 / 23	83.3 %	5 / 6
cpp_tlc5955/inc/tlc5955.hpp		100.0 %	1 / 1	- %	0 / 0
cpp_tlc5955/src/tlc5955.cpp		96.1 %	171 / 178	55.6 %	174 / 313
main_app/src/mainapp.cpp		0.0 %	0 / 29	0.0 %	0 / 42

Generated by: GCOVR (Version 4.2)

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
File:	cpp_ssd1306/inc/font.hpp	Lines:	8	8	100.0 %
Date:	2021-12-05 16:22:01	Branches:	2	2	100.0 %

```
Line Branch  Exec Source
   1
   2
   3              #ifndef __FONT_HPP__
   4              #define __FONT_HPP__
   5
   6              #include <stdint.h>
   7              #include <array>
   8              //#include <variant>
   9              //#include <fontdata.hpp>
  10
  11
  12
  13              namespace ssd1306
  14              {
  15
  16              template<std::size_t FONT_SIZE>
  17              class Font
  18              {
  19
  20              public:
  21
  22               // @brief Construct a new Font object
  23               Font() = default;
  24
  25               // @brief function to get a font pixel (16bit half-word).
  26               // @param idx The position in the font data array to retrieve data
  27               // @return uint16_t The halfword of data we retrieve
  28          522  bool get_pixel(size_t idx, uint32_t &bit_line)
  29               {
  30     ✓✓   522   if (idx > data.size())
  31                {
  32            2    return false;
  33                }
  34                else
  35                {
  36          520    bit_line = static_cast<uint32_t>(data.at(idx));
  37          520    return true;
  38                }
  39               }
  40
  41               // @brief get the width member variable
  42               // @return uint8_t the width value
  43         8869  uint8_t width() { return m_width; }
  44
  45               // @brief get tte height member variable
  46               // @return uint8_t the height value
  47         1711  uint8_t height() { return m_height; }
  48
  49               // @brief helper function to get the size of the private font data array.
  50               // @return size_t the array size
  51           10  size_t size() { return data.size(); }
  52
  53               std::array<char, 95> character_map {
  54                ' ', '!', '"', '#', '$', '%', '&', '\'','(', ')',
  55                '*', '+', ',', '-', '.', '/', '0', '1', '2', '3',
  56                '4', '5', '6', '7', '8', '9', ':', ';', '<', '=',
  57                '>', '?', '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
  58                'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q',
  59                'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '[',
  60                '\\',']', '^', '_', '`', 'a', 'b', 'c', 'd', 'e',
  61                'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
  62                'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y',
  63                'z', '{', '|', '}', '~'
  64               };
  65
  66              private:
  67
```

```
 // @brief The width of the font in pixels
 static uint8_t m_width;

 // @brief The height of the font in pixels
 static uint8_t m_height;

 // @brief the font data
 static std::array<uint16_t, FONT_SIZE> data;

};

// specializations
typedef Font<475> Font5x5;
typedef Font<680> Font5x7;
typedef Font<950> Font7x10;
typedef Font<1710> Font11x18;
typedef Font<2470> Font16x26;

} // namespace ssd1306

#endif // __FONT_HPP__
```

Generated by: GCOVR (Version 4.2)

GCC Code Coverage Report

Directory:	./		Exec	Total	Coverage
File:	cpp_ssd1306/inc/ssd1306.hpp	Lines:	43	50	86.0 %
Date:	2021-12-05 16:22:01	Branches:	9	62	14.5 %

```
LineBranch Exec  Source
  1                /*
  2                 * Display.hpp
  3                 *
  4                 *  Created on: 7 Nov 2021
  5                 *      Author: chris
  6                 */
  7
  8                // @note See datasheet
  9                // https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf
 10
 11                #ifndef Display_HPP_
 12                #define Display_HPP_
 13
 14                #include <variant>
 15                #include <font.hpp>
 16                #include <sstream>
 17                #include <iostream>
 18                #include <array>
 19                #include <utility>
 20
 21
 22
 23                #ifdef USE_HAL_DRIVER
 24                 #include "stm32g0xx.h"
 25                 #include "main.h"
 26                 #include "spi.h"
 27                #endif
 28
 29
 30                namespace ssd1306
 31                {
 32
 33                // @brief
 34                enum class Colour: uint16_t
 35                {
 36                    Black = 0x00,
 37                    White = 0x01
 38                };
 39
 40                // @brief
 41                class Display
 42                {
 43                public:
 44             5   Display() = default;
 45
 46            10   virtual ~Display() = default;
 47
 48                 // @brief
 49                 bool init();
 50
 51
 52                 // @brief
 53                 // @tparam FONT_SIZE
 54                 // @param msg
 55                 // @param font
 56                 // @param x
 57                 // @param y
 58                 // @param bg
 59                 // @param fg
 60                 // @param padding
 61                 // @param update
 62                 // @return char
 63                 template<std::size_t FONT_SIZE>
 64                 char write(std::stringstream &msg, Font<FONT_SIZE> &font, uint8_t x, uint8_t y, Colour bg, Colour fg, bool padding, bool update);
 65
 66
 67                 // @brief Get the display width. Can be used to create a std::array
 68                 // @return constexpr uint16_t
 69                 static constexpr uint16_t get_display_width() { return m_width; }
 70
 71                 // @brief Get the display height. Can be used to create a std::array
 72                 // @return constexpr uint16_t
 73                 static constexpr uint16_t get_display_height() { return m_height; }
 74
 75                private:
 76
 77                 // @brief
 78                 // @param x
 79                 // @param y
 80                 // @param colour
 81                 bool draw_pixel(uint8_t x, uint8_t y, Colour colour);
 82
 83                 // @brief
 84                 // @param colour
 85                 void fill(Colour colour);
 86
 87                 // @brief
 88                 bool update_screen();
 89
 90                 // @brief
 91                 void reset();
 92
 93                 // @brief Set the cursor object
 94                 // @param x
```

```
          // @param y
          bool set_cursor(uint8_t x, uint8_t y);


          // @brief
          // @param cmd_byte
          bool write_command(uint8_t cmd_byte);

          // @brief
          // @param data_buffer
          // @param data_buffer_size
          bool write_data(uint8_t* data_buffer, size_t data_buffer_size);

          // @brief
             uint16_t m_currentx {0};

          // @brief
             uint16_t m_currenty {0};

          // @brief
             uint8_t m_inverted {0};

          // @brief
             uint8_t m_initialized {0};

          // @brief The display width in bytes. Used in std::array.
             static const uint16_t m_width {128};

          // @brief The display height, in bytes. Used in std::array.
             static const uint16_t m_height {64};

         #ifdef USE_HAL_DRIVER

          // @brief
          SPI_HandleTypeDef m_spi_port {hspi1};
          // @brief
          uint16_t m_cs_port {0};
          // @brief
          uint16_t m_cs_pin {0};
          // @brief
          GPIO_TypeDef* m_dc_port {SPI1_DC_GPIO_Port};
          // @brief
          uint16_t m_dc_pin {SPI1_DC_Pin};
          // @brief
          GPIO_TypeDef* m_reset_port {SPI1_RESET_GPIO_Port};
          // @brief
          uint16_t m_reset_pin {SPI1_RESET_Pin};

         #endif

         protected:


          // @brief byte buffer for ssd1306. Access to derived classes like ssd1306_tester is permitted.
             std::array<uint8_t, (m_width*m_height)/8> m_buffer;

          // @brief
          // @tparam FONT_SIZE
          // @param ss
          // @param font
          // @param colour
          // @param padding
          // @return char
          template<std::size_t FONT_SIZE>
          char write_string(std::stringstream &ss, Font<FONT_SIZE> &font, Colour colour, bool padding);

          // @brief
          // @tparam FONT_SIZE
          // @param ch
          // @param font
          // @param colour
          // @param padding
          // @return char
          template<std::size_t FONT_SIZE>
          char write_char(char ch, Font<FONT_SIZE> &font, Colour colour, bool padding);


          // @brief Get the buffer object. Used for testing only.
          // @notes use
          // @param buffer
          //void get_buffer(std::array<uint8_t, (m_width*m_height)/8> &buffer) { buffer = m_buffer; }

         };

         // Out-of-class definitions of member function templates

         template<std::size_t FONT_SIZE>
      12 char Display::write(std::stringstream &msg, Font<FONT_SIZE> &font, uint8_t x, uint8_t y, Colour bg, Colour fg, bool padding, bool update)
         {

      12     fill(bg);
✓✓    12     if (!set_cursor(x, y))
          {
       4   return 0;
          }
       8     char res = write_string(msg, font, fg, padding);
✓✗     8     if (update)
             {
       8         update_screen();
             }
       8     return res;
         }

```

```
                template<std::size_t FONT_SIZE>
            30  char Display::write_string(std::stringstream &ss, Font<FONT_SIZE> &font, Colour color, bool padding)
                {
                    // Write until null-byte
                 char ch;
✓✗✓✗
    ✓✓      30      while (ss.get(ch))
                    {
✓✗✗✓        20          if (write_char(ch, font, color, padding) != ch)
                        {
                            // Char could not be written
                            return ch;
                        }
                    }

                    // Everything ok
            10      return ch;
                }

                template<std::size_t FONT_SIZE>
            11  char Display::write_char(char ch, Font<FONT_SIZE> &font, Colour colour, bool padding)
                {

                    // Check remaining space on current line
✗✗✗✗        22      if (m_width <= (m_currentx + font.height()) ||
  ✗✗        11          m_width <= (m_currenty + font.height()))
                    {
                        // Not enough space on current line
                        return 0;
                    }

                    // add extra leading horizontal space
  ✗✗        11      if (padding)
                    {
  ✗✗       297       for(size_t n = 0; n < font.height(); n++)
                  {
✗✗✗✗       286     if (!draw_pixel(m_currentx, (m_currenty + n), Colour::Black))
                   {
                    return false;
                   }
                  }
            11       m_currentx += 1;
                    }

                    // Use the font to write
                    uint32_t font_data_word;
  ✗✗       271      for(size_t font_height_idx = 0; font_height_idx < font.height(); font_height_idx++)
                 {
✗✗✗✗       261          if (!font.get_pixel( (ch - 32) * font.height() + font_height_idx, font_data_word )) { return false; }

                #ifdef ENABLE_SSD1306_TEST_STDOUT
                  // separator for the font
                        std::cout << std::endl;
                #endif

  ✗✗      4420          for(size_t font_width_idx = 0; font_width_idx < font.width(); font_width_idx++)
                  {
  ✗✗      4160              if((font_data_word << font_width_idx) & 0x8000)
                            {
 ✗✗✗      1610               switch (colour)
                    {
          1117       case Colour::White:
✗✗✗✗      1117        if (!draw_pixel(m_currentx + font_width_idx, m_currenty + font_height_idx, Colour::White))
                      {
                       return false;
                      }
          1117        break;

           493       case Colour::Black:
✗✗✗✗       493        if (!draw_pixel(m_currentx + font_width_idx, m_currenty + font_height_idx, Colour::Black))
                      {
                       return false;
                      }
           493        break;
                    }
                            }
                            else
                            {
 ✗✗✗      2550               switch (colour)
                    {
          1379       case Colour::White:
✗✗✗✗      1379        if (!draw_pixel(m_currentx + font_width_idx, m_currenty + font_height_idx, Colour::Black))
                      {
                       return false;
                      }
          1379        break;

          1171       case Colour::Black:
✗✗✗✗      1171        if (!draw_pixel(m_currentx + font_width_idx, m_currenty + font_height_idx, Colour::White))
                      {
                       return false;
                      }
          1171        break;
                    }
                            }
                        }
                    }

                    // The current space is now taken
            10      m_currentx += font.width();

                    // add extra leading horizontal space
298  ✗✗  10      if (padding)
299            {
300      10       m_currentx += 1;
301            }
302
303              // Return written char for validation
304      10      return ch;
305          }
306
307
308
309          } // namespace ssd1306
310
311          #endif /* Display_HPP_ */
```

Generated by: GCOVR (Version 4.2)

GCC Code Coverage Report

				Exec	Total	Coverage
Directory:	./					
File:	cpp_ssd1306/src/ssd1306.cpp	Lines:		65	65	100.0 %
Date:	2021-12-05 16:22:01	Branches:		44	76	57.9 %

```
Line Branch  Exec  Source
   1                /*
   2                 * Display.cpp
   3                 *
   4                 *  Created on: 7 Nov 2021
   5                 *      Author: chris
   6                 */
   7
   8                // @note See datasheet
   9                // https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf
  10
  11                #include "ssd1306.hpp"
  12                #include <iomanip>
  13                #include <bitset>
  14
  15                namespace ssd1306
  16                {
  17
  18                8  bool Display::init()
  19                {
  20                8      bool res = true;
  21                 // Reset Display
  22                8   reset();
  23
  24                    // Wait for the screen to boot
  25                #ifdef USE_HAL_DRIVER
  26                    HAL_Delay(100);
  27                #endif
  28                    // Init Display
  29     ✗✓      8      if (!write_command(0xAE)) { return false; } //display off
  30     ✗✓      8      if (!write_command(0x20)) { return false; } //Set Memory Addressing Mode
  31     ✗✓      8      if (!write_command(0x10)) { return false; } // 00,Horizontal Addressing Mode; 01,Vertical Addressing Mode; 10,Page Addressing Mode (RESET); 11,Invalid
  32     ✗✓      8      if (!write_command(0xB0)) { return false; } //Set Page Start Address for Page Addressing Mode,0-7
  33     ✗✓      8      if (!write_command(0xC8)) { return false; } //Set COM Output Scan Direction
  34     ✗✓      8      if (!write_command(0x00)) { return false; } //---set low column address
  35     ✗✓      8      if (!write_command(0x10)) { return false; } //---set high column address
  36     ✗✓      8      if (!write_command(0x40)) { return false; } //--set start line address - CHECK
  37     ✗✓      8      if (!write_command(0x81)) { return false; } //--set contrast control register - CHECK
  38     ✗✓      8      if (!write_command(0xFF)) { return false; }
  39     ✗✓      8      if (!write_command(0xA1)) { return false; } //--set segment re-map 0 to 127 - CHECK
  40     ✗✓      8      if (!write_command(0xA6)) { return false; } //--set normal color
  41     ✗✓      8      if (!write_command(0xA8)) { return false; } //--set multiplex ratio(1 to 64) - CHECK
  42     ✗✓      8      if (!write_command(0x3F)) { return false; } //
  43     ✗✓      8      if (!write_command(0xA4)) { return false; } //0xa4,Output follows RAM content;0xa5,Output ignores RAM content
  44     ✗✓      8      if (!write_command(0xD3)) { return false; } //-set display offset - CHECK
  45     ✗✓      8      if (!write_command(0x00)) { return false; } //-not offset
  46     ✗✓      8      if (!write_command(0xD5)) { return false; } //--set display clock divide ratio/oscillator frequency
  47     ✗✓      8      if (!write_command(0xF0)) { return false; } //--set divide ratio
  48     ✗✓      8      if (!write_command(0xD9)) { return false; } //--set pre-charge period
  49     ✗✓      8      if (!write_command(0x22)) { return false; } //
  50     ✗✓      8      if (!write_command(0xDA)) { return false; } //--set com pins hardware configuration - CHECK
  51     ✗✓      8      if (!write_command(0x12)) { return false; }
  52     ✗✓      8      if (!write_command(0xDB)) { return false; } //--set vcomh
  53     ✗✓      8      if (!write_command(0x20)) { return false; } //0x20,0.77xVcc
  54     ✗✓      8      if (!write_command(0x8D)) { return false; } //--set DC-DC enable
  55     ✗✓      8      if (!write_command(0x14)) { return false; } //
  56     ✗✓      8      if (!write_command(0xAF)) { return false; } //--turn on Display panel
  57
  58                    // Clear screen
  59             8      fill(Colour::Black);
  60
  61                    // Flush buffer to screen
  62             8      update_screen();
  63
  64                    // Set default values for screen object
  65             8      m_currentx = 0;
  66             8      m_currenty = 0;
  67
  68             8      m_initialized = 1;
  69
  70             8      return res;
  71                }
  72
  73
  74            14  void Display::fill(Colour color)
  75                {
  76     ✓✓  14350      for(auto &pixel : m_buffer)
  77                    {
  78     ✓✓  14336          pixel = (color == Colour::Black) ? 0x00 : 0xFF;
  79                    }
  80            14  }
  81
  82            12  bool Display::update_screen()
  83                {
  84     ✓✓    108      for(uint8_t i = 0; i < 8; i++)
  85                    {
  86     ✗✓     96          if (!write_command(0xB0 + i)) { return false; }
  87     ✗✓     96          if (!write_command(0x00)) { return false; }
  88     ✗✓     96          if (!write_command(0x10)) { return false; }
  89     ✗✓     96          if (!write_data(&m_buffer[m_width * i], m_width)) { return false; }
  90                    }
  91            12      return true;
  92                }
  93
  94          4446  bool Display::draw_pixel(uint8_t x, uint8_t y, Colour color)
  95                {
  96                    // Draw in the right color
  97     ✓✓   4446      if(color == Colour::White)
  98                    {
  99          2288          m_buffer[x + (y / 8) * m_width] |= 1 << (y % 8);
 100                #ifdef ENABLE_SSD1306_TEST_STDOUT
 101                        std::cout << "1";
 102                #endif
 103                    }
 104                    else
```

```
                {
        2158        m_buffer[x + (y / 8) * m_width] &= ~(1 << (y % 8));
            #ifdef ENABLE_SSD1306_TEST_STDOUT
                    std::cout << "_";
            #endif
                }

        4446    return true;
            }

           6 bool Display::set_cursor(uint8_t x, uint8_t y)
            {
✓✓✓✓       6    if(x >= m_width || y >= m_height)
                {
           2        return false;
                }
                else
                {
           4        m_currentx = x;
           4        m_currenty = y;
                }
           4    return true;
            }


           8 void Display::reset()
            {
             // CS = High (not selected)
             //HAL_GPIO_WritePin(Display_CS_Port, Display_CS_Pin, GPIO_PIN_SET);

             // Reset the Display
            #ifdef USE_HAL_DRIVER
             HAL_GPIO_WritePin(m_reset_port, m_reset_pin, GPIO_PIN_RESET);
             HAL_Delay(10);
             HAL_GPIO_WritePin(m_reset_port, m_reset_pin, GPIO_PIN_SET);
             HAL_Delay(10);
            #endif
           8 }

         512 bool Display::write_command(uint8_t cmd_byte __attribute__((unused)))
            {
            #ifdef USE_HAL_DRIVER
                HAL_StatusTypeDef res = HAL_OK;
             //HAL_GPIO_WritePin(m_cs_port, m_cs_pin, GPIO_PIN_RESET); // select Display
             HAL_GPIO_WritePin(m_dc_port, m_dc_pin, GPIO_PIN_RESET); // command
             res = HAL_SPI_Transmit(&m_spi_port, (uint8_t *) &cmd_byte, 1, HAL_MAX_DELAY);
                if (res != HAL_OK)
                {
                    return false;
                }
                return true;
             //HAL_GPIO_WritePin(m_cs_port, m_cs_pin, GPIO_PIN_SET); // un-select Display
            #else
         512    return true;
            #endif
            }

          96 bool Display::write_data(uint8_t* data_buffer __attribute__((unused)), size_t data_buffer_size __attribute__((unused)))
            {
            #ifdef USE_HAL_DRIVER
                HAL_StatusTypeDef res = HAL_OK;
             //HAL_GPIO_WritePin(m_cs_port, m_cs_pin, GPIO_PIN_RESET); // select Display
             HAL_GPIO_WritePin(m_dc_port, m_dc_pin, GPIO_PIN_SET); // data
             res = HAL_SPI_Transmit(&m_spi_port, data_buffer, data_buffer_size, HAL_MAX_DELAY);
                if (res != HAL_OK)
                {
                    return false;
                }
                return true;
             //HAL_GPIO_WritePin(m_cs_port, m_cs_pin, GPIO_PIN_SET); // un-select Display
            #else
          96    return true;
            #endif



            }

            } // namespace ssd1306
```

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
File:	cpp_tlc5955/inc/byte_position.hpp	Lines:	23	23	100.0 %
Date:	2021-12-05 16:22:01	Branches:	5	6	83.3 %

```
Line Branch Exec Source
   1             #include <cstdio>
   2             #include <cstdint>
   3             #include <iostream>
   4             #include <array>
   5             #include <limits>
   6
   7             #include <cstdint>
   8             #ifndef STM32G0B1xx
   9                 #include <stdexcept>
  10                 #define STDOUT_DEBUG
  11             #endif
  12
  13
  14             // @brief Manages byte count and bit position index. Bit position is reset everytime the byte position is incremented.
  15             class BytePosition
  16             {
  17             public:
  18
  19                 // @brief Construct a new Byte Position object
  20                 // @param init_byte_pos The byte counter index
  21                 // @param init_bit_idx The max bit position within the byte.
  22              72     BytePosition(const uint16_t init_byte_pos, const uint16_t init_bit_idx = 8)
  23              72         :   m_byte_position(init_byte_pos),
  24              72             m_max_bit_idx (init_bit_idx)
  25                 {
  26              72     };
  27
  28                 // @brief pre-increment the byte position. Implicitly resets bit position to max_bit_idx.
  29                 // @return uint16_t
  30                 uint16_t operator++()
  31                 {
  32                     //m_bit_position = m_max_bit_idx;
  33                     //return m_byte_position++;
  34                     return m_byte_position;
  35                 }
  36
  37                 // @brief post-increment the byte position. Implicitly resets bit position to max_bit_idx.
  38                 // @return uint16_t&
  39               7     uint16_t& operator++(const int)
  40                 {
  41                     //m_bit_position = m_max_bit_idx;
  42                     //return ++m_byte_position;
  43               7         return m_byte_position;
  44                 }
  45
  46              32     void set_byte_idx(const uint16_t position)
  47                 {
  48              32         m_byte_position = position;
  49              32     }
  50
  51            2673     uint16_t get_byte_idx() const
  52                 {
  53            2673         return m_byte_position;
  54                 }
  55
  56
  57
  58                 // @brief check if we reach end of byte without decrementing the bit position.
  59                 // @return true if there are more bits in this byte. false if end of byte is reached.
  60                 bool has_next() {
  61                     if (m_bit_position == std::numeric_limits<uint16_t>::max())
  62                     {
  63                         return false;
  64                     }
  65                     return true;
  66                 }
  67
  68                 // @brief get the bit index and post-decrement it.

  69                 // Throws exception (x86) or returns zero (Arm) if next bit pos == std::numeric_limits<uint16_t>::max()
  70                 // @return uint16_t The bit position value.
  71            2825     uint16_t next_bit_idx()
  72                 {
  73
  74
  75            2825         --m_bit_position;
  76
  77        ✓✓  2825         if (m_bit_position == std::numeric_limits<uint16_t>::max())
  78                     {
  79                         // reset the bit position back to the max value
  80             329             m_bit_position = m_max_bit_idx;
  81             329             ++m_byte_position;
```

```
    329             --m_bit_position;
                }

                #ifdef STDOUT_DEBUG
   2825             std::cout << "Byte:" << m_byte_position << " Bit:" << m_bit_position << std::endl;
                #endif


   2825         return m_bit_position;
            }

            // @brief Convenience function. Calls next_bit_idx() function N times.
            // @param n The number of bits to skip forward
            // @return true if next_bit_idx() is successful, false if not
     60     bool skip_next_n_bits(uint8_t n)
            {
✓✓  208         for (uint8_t idx = 0; idx < n; idx++)
                {
✗✓  148             if (!next_bit_idx()) { return false; }
                }
     60         return true;
            }

        private:

            uint16_t m_byte_position {0};
            uint16_t m_max_bit_idx {0};
            uint16_t m_bit_position {m_max_bit_idx};

        };
```

Generated by: GCOVR (Version 4.2)

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
File:	cpp_tlc5955/inc/tlc5955.hpp	Lines:	1	1	100.0 %
Date:	2021-12-05 16:22:01	Branches:	0	0	- %

```
Line Branch Exec  Source
   1
   2              #include <stdint.h>
   3              #include <bitset>
   4
   5              #ifdef USE_HAL_DRIVER
   6                  #include "stm32g0xx.h"
   7                  #include "main.h"
   8              #endif
   9
  10
  11              //#include "spi.h"
  12
  13              #include <ssd1306.hpp>
  14
  15              namespace tlc5955 {
  16
  17              // https://godbolt.org/z/1q9sn3Gar
  18
  19              class Driver
  20              {
  21              public:
  22
  23                  Driver() = default;
  24
  25             54      virtual ~Driver() = default;
  26
  27                  static const uint8_t m_bc_data_resolution {7};
  28                  static const uint8_t m_mc_data_resolution {3};
  29                  static const uint8_t m_dc_data_resolution {7};
  30                  static const uint8_t m_gs_data_resolution {16};
  31
  32                  void set_control_bit(bool ctrl);
  33
  34                  void set_ctrl_cmd_bits();
  35
  36                  void set_padding_bits();
  37
  38                  // @brief Set the Function Control (FC) data latch.
  39                  // See section 8.3.2.7 "Function Control (FC) Data Latch" (page 23).
  40                  // https://www.ti.com/lit/ds/symlink/tlc5955.pdf
  41                  // @param DSPRPT Auto display repeat mode enable bit. When enabled each output repeats the PWM control every 65,536 GSCLKs.
  42                  // @param TMGRST Display timing reset mode enable bit. When enabled the GS counter resets and outputs forced off at the latch rising edge
  43                  // for a GS data write
  44                  // @param RFRESH Data in the common register are copied to the GS data latch and DC data in the control data latch are copied to the DC data latch
  45                  // at the 65,536th GSCLK after the LAT rising edge for a GS data write.
  46                  // @param ESPWM When 0, conventional PWM is selected. When 1, Enhanced Spectrum (ES) PWM is selected. See 8.4.4 "Grayscale (GS) Function (PWM Control)"
  47                  // @param LSDVLT LED short detection (LSD) detection voltage selection bit. When this bit is 0, the LSD voltage is VCC × 70%.
  48                  // When this bit is 1, the LSD voltage is VCC × 90%. See 8.3.5 "LED Short Detection (LSD)"
  49                  void set_function_data(bool DSPRPT, bool TMGRST, bool RFRESH, bool ESPWM, bool LSDVLT);
  50
  51                  // @brief Write the Global BC (Bright Control) data to the common register.
  52                  // https://www.ti.com/lit/ds/symlink/tlc5955.pdf
  53                  // @param blue_value The 7-bit word for blue BC
  54                  // @param green_value The 7-bit word for green BC
  55                  // @param red_value The 7-bit word for red BC
  56                  void set_bc_data(
  57                      const std::bitset<m_bc_data_resolution> &blue_value,
  58                      const std::bitset<m_bc_data_resolution> &green_value,
  59                      const std::bitset<m_bc_data_resolution> &red_value);
  60
  61                  // @brief Write the MC (Max Current) data to the common register
  62                  // https://www.ti.com/lit/ds/symlink/tlc5955.pdf
  63                  // @param blue_value The 3-bit word for blue MC
  64                  // @param green_value The 3-bit word for green MC
  65                  // @param red_value The 3-bit word for red MC
  66                  void set_mc_data(
  67                      const std::bitset<m_mc_data_resolution> &blue_value,
  68                      const std::bitset<m_mc_data_resolution> green_value,

  69                      const std::bitset<m_mc_data_resolution> &red_value);
  70
  71                  // @brief Write the DC (dot correction) data to the common register for the specified LED
  72                  // https://www.ti.com/lit/ds/symlink/tlc5955.pdf
  73                  // @param led_idx The selected LED
  74                  // @param blue_value The 7-bit word for blue DC
  75                  // @param green_value The 7-bit word for green DC
  76                  // @param red_value The 7-bit word for red DC
  77                  bool set_dc_data(
  78                      uint8_t led_idx,
  79                      const std::bitset<m_dc_data_resolution> &blue_value,
  80                      const std::bitset<m_dc_data_resolution> &green_value,
  81                      const std::bitset<m_dc_data_resolution> &red_value);
  82
  83                  // @brief Convenience function to set all LEDs to the same DC values
  84                  // @param blue_value The 7-bit word for blue DC
  85                  // @param green_value The 7-bit word for green DC
  86                  // @param red_value The 7-bit word for red DC
  87                  void set_all_dc_data(
  88                      const std::bitset<m_dc_data_resolution> &blue_value,
  89                      const std::bitset<m_dc_data_resolution> &green_value,
  90                      const std::bitset<m_dc_data_resolution> &red_value);
  91
  92                  // @brief Write the GS (Grey Scale) data to the common register for the specified LED
  93                  // @param led_pos The selected LED
  94                  // @param blue_value The 16-bit word for blue GS
  95                  // @param green_value The 16-bit word for green GS
  96                  // @param red_value The 16-bit word for red GS
  97                  bool set_gs_data(
  98                      uint8_t led_idx,
  99                      const std::bitset<m_gs_data_resolution> &blue_value,
 100                      const std::bitset<m_gs_data_resolution> &green_value,
 101                      const std::bitset<m_gs_data_resolution> &red_value);
 102
 103                  // @brief Convenience function to set all LEDs to the same GS values
 104                  // @param blue_value The 16-bit word for blue GS
 105                  // @param green_value The 16-bit word for green GS
 106                  // @param red_value The 16-bit word for red GS
 107                  void set_all_gs_data(
 108                      const std::bitset<m_gs_data_resolution> &blue_value,
 109                      const std::bitset<m_gs_data_resolution> &green_value,
 110                      const std::bitset<m_gs_data_resolution> &red_value);
 111
 112                  // @brief Send the data via SPI bus and toggle the latch pin
 113                  void send_data();
```

```

    // @brief Clears (zeroize) the common register and call send_data()
    void flush_common_register();

    // @brief toggle the latch pin terminal
    void toggle_latch();

    // @brief Helper function to print bytes as decimal values to RTT. USE_RTT must be defined.
    void print_common_bits();

    // @brief sections offsets for common register
    enum byte_offsets {
        // @brief 1 for control data latch, 0 for greyscale data latch
        latch = 0U,
        // @brief Used in control data latch.
        ctrl_cmd = 0U,
        // @brief Used in greyscale data latch.
        greyscale = 1U,
        // @brief Used in control data latch. Don't care bits.
        padding = 1U,
        // @brief Used in control data latch.
        function = 49U,
        // @brief Used in control data latch.
        brightness_control = 50U,
        // @brief Used in control data latch.
        max_current = 53U,
        // @brief Used in control data latch.
        dot_correct = 54U
    };



protected:

    static const uint8_t m_common_reg_size_bytes {97};
    std::array<uint8_t, m_common_reg_size_bytes> m_common_byte_register{0};

private:

    uint8_t built_in_test_fail {0};

    // Bits required for correct control reg size
    static const uint16_t m_common_reg_size_bits {769};


     // @brief The number of daisy chained driver chips in the circuit.
    uint8_t m_num_driver_ics {1};

    // @brief The number of colour channels per LED
    static const uint8_t m_num_colour_chan {3};

    // @brief The number of LEDs per driver chip
    static const uint8_t m_num_leds_per_chip {16};



    // the size of each common register section
    static const uint8_t m_latch_size_bits {1};                                                                             // 1U
    static const uint8_t m_ctrl_cmd_size_bits {8};                                                                          // 8U
    static constexpr uint16_t m_gs_data_one_led_size_bits {m_gs_data_resolution * m_num_colour_chan};                       // 48U
    static constexpr uint16_t m_gs_data_section_size_bits {m_gs_data_resolution * m_num_leds_per_chip * m_num_colour_chan}; // 768U
    static const uint8_t m_func_data_section_size_bits {5};                                                                 // 5U
    static constexpr uint8_t m_bc_data_section_size_bits {m_bc_data_resolution * m_num_colour_chan};                        // 21U
    static constexpr uint8_t m_mc_data_section_size_bits {m_mc_data_resolution * m_num_colour_chan};                        // 9U
    static constexpr uint8_t m_dc_data_one_led_size_bits {m_dc_data_resolution * m_num_colour_chan};                        // 21U
    static constexpr uint16_t m_dc_data_section_size_bits {m_dc_data_resolution * m_num_leds_per_chip * m_num_colour_chan}; // 336U
    static constexpr uint16_t m_padding_section_size_bits {                                                                 // 389U
        m_common_reg_size_bits  - m_latch_size_bits - m_ctrl_cmd_size_bits - m_func_data_section_size_bits - m_bc_data_section_size_bits - m_mc_data_section_size_bits
    };

    // the offset of each common register section
    // static const uint8_t m_latch_offset {0};
    // static constexpr uint8_t m_ctrl_cmd_offset {static_cast<uint8_t>(m_latch_offset + m_latch_size_bits)};                  // 1U
    // static constexpr uint8_t m_gs_data_offset {static_cast<uint8_t>(m_ctrl_cmd_offset)};                                    // 9U - used in gs data latch only
    // static constexpr uint8_t m_padding_offset {static_cast<uint8_t>(m_ctrl_cmd_offset + m_ctrl_cmd_size_bits)};             // 9U - used in ctrl data latch only
    // static constexpr uint16_t m_func_data_offset {static_cast<uint16_t>(m_padding_offset + m_padding_section_size_bits)};   // 9U
    // static constexpr uint16_t m_bc_data_offset {static_cast<uint16_t>(m_func_data_offset + m_func_data_section_size_bits)}; // 398U
    // static constexpr uint16_t m_mc_data_offset {static_cast<uint16_t>(m_bc_data_offset + m_bc_data_section_size_bits)};     // 424U
    // static constexpr uint16_t m_dc_data_offset {static_cast<uint16_t>(m_mc_data_offset + m_mc_data_section_size_bits)};     // 433U


    // @brief Helper function to set/clear one bit of one byte in the common register byte array
    // @param byte The targetted byte in the common register
    // @param bit The bit within that byte to be set/cleared
    // @param _new_value The boolean value to set at the bit target_idx
    void set_value_nth_bit(uint8_t &byte, uint16_t bit, bool new_value);


    std::bitset<m_common_reg_size_bits> m_common_bit_register{0};

    const uint8_t  m_latch_delay_ms {1};

    // @brief Predefined write command.
    // section 8.3.2.3 "Control Data Latch" (page 21).
    // section 8.3.2.2 "Grayscale (GS) Data Latch" (page 20).
    // https://www.ti.com/lit/ds/symlink/tlc5955.pdf
    std::bitset<8> m_ctrl_cmd {0x96};

    // @brief Predefined flush command
    std::bitset<8> m_flush_cmd {0};

    // void enable_spi();
    // void disable_spi();

    // void enable_gpio_output_only();
#ifdef USE_HAL_DRIVER
    // @brief The HAL SPI interface
    SPI_HandleTypeDef m_spi_interface {hspi2};
    // @brief Latch GPIO pin
 uint16_t m_lat_pin {TLC5955_SPI2_LAT_Pin};
    // @brief Latch terminal GPIO port
    GPIO_TypeDef* m_lat_port {TLC5955_SPI2_LAT_GPIO_Port};
    // @brief GreyScale clock GPIO pin
    uint16_t m_gsclk_pin {TLC5955_SPI2_GSCLK_Pin};
    // @brief GreyScale clock GPIO port
    GPIO_TypeDef* m_gsclk_port {TLC5955_SPI2_GSCLK_GPIO_Port};
    // @brief SPI MOSI GPIO pin
    uint16_t m_mosi_pin {TLC5955_SPI2_MOSI_Pin};
    // @brief SPI MOSI GPIO port
    GPIO_TypeDef* m_mosi_port {TLC5955_SPI2_MOSI_GPIO_Port};
    // @brief SPI Clock GPIO pin
    uint16_t m_sck_pin {TLC5955_SPI2_SCK_Pin};
    // @brief SPI Clock GPIO port
    GPIO_TypeDef* m_sck_port {TLC5955_SPI2_SCK_GPIO_Port};
#endif

};

} // tlc5955
```

Generated by: GCOVR (Version 4.2)

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
File:	cpp_tlc5955/src/tlc5955.cpp	Lines:	171	178	96.1 %
Date:	2021-12-05 16:22:01	Branches:	174	313	55.6 %

```
Line Branch Exec  Source
   1
   2                #include "tlc5955.hpp"
   3                #include <sstream>
   4                #include <cmath>
   5                #include <cstring>
   6                #include <byte_position.hpp>
   7
   8                #ifdef USE_RTT
   9                    #include <SEGGER_RTT.h>
  10                #endif
  11                namespace tlc5955
  12                {
  13
  14
  15         2690   void Driver::set_value_nth_bit(uint8_t &byte, uint16_t bit, bool new_value)
  16                {
  17     ✓✓  2690       if (new_value) { byte |= (1U << bit); }
  18         1536       else { byte &= ~(1U << bit); }
  19         2690       print_common_bits();
  20         2690   }
  21
  22
  23            1   void Driver::set_control_bit(bool ctrl_latch)
  24                {
  25                    // Latch
  26                    // bits      =
  27                    // Bytes     [
  28                    //          #0
  29
  30                    //m_common_bit_register.set(m_latch_offset, ctrl_latch);
  31            1       set_value_nth_bit(m_common_byte_register[byte_offsets::latch], 7, ctrl_latch);
  32            1   }
  33
  34            1   void Driver::set_ctrl_cmd_bits()
  35                {
  36
  37                    // Ctrl      10010110
  38                    // bits      [======]
  39                    // Bytes     ======][
  40                    //             #0    #1
  41
  42                    // BYTE #0
  43            1       BytePosition byte_pos(byte_offsets::ctrl_cmd);
  44
  45                    // skip the MSB of byte #0
  46     ✓✗     1       byte_pos.next_bit_idx();
  47
  48                    // 7 MSB bits of ctrl byte into 7 LSB of byte #0
  49     ✓✓     8       for (int8_t idx = m_ctrl_cmd_size_bits - 1; idx > 0; idx--)
  50                    {
  51   ✓✗✓✗     7           set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx() , m_ctrl_cmd.test(idx));
         ✓✗
  52                    }
  53
  54                    // the last m_ctrl_cmd bit in to MSB of byte #1
  55            1       byte_pos++;
  56   ✓✗✓✗     1       set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), m_ctrl_cmd.test(0));
         ✓✗
  57
  58            1   }
  59
  60            1   void Driver::set_padding_bits()
  61                {
  62
  63                    // Padding  0  ================================================================================ 79
  64                    // Bytes       ======][======][======][======][======][======][======][======][======][======][
  65                    //               #1      #2       #3    #4      #5      #6      #7      #8      #9     #10
  66
  67                    // Padding 80  ================================================================================ 159
  68                    // Bytes       ======][======][======][======][======][======][======][======][======][======][
  69                    //              #11     #12     #13     #14     #15     #16     #17     #18     #19     #20
  70
  71                    // Padding 160 ================================================================================ 239
  72                    // Bytes       ======][======][======][======][======][======][======][======][======][======][
  73                    //              #21     #22     #23     #24     #25     #26     #27     #28     #29     #30
  74
  75                    // Padding 240 ================================================================================ 319
  76                    // Bytes       ======][======][======][======][======][======][======][======][======][======][
  77                    //              #31     #32     #33     #34     #35     #36     #37     #38     #39     #40
  78
  79                    // Padding 320 ===================================================================== 389
  80                    // Bytes       ======][======][======][======][======][======][======][======][=====
  81                    //              #41     #42     #43     #44     #45     #46     #47     #48     #49
  82
  83                    // BYTE #1
  84            1       BytePosition byte_pos(byte_offsets::padding);
  85
  86                    // skip MSB of byte #1
  87     ✓✗     1       byte_pos.next_bit_idx();
  88
  89                    // then write next 7 LSB bits of byte #1
  90     ✓✓   389       for (uint16_t idx = 0; idx < m_padding_section_size_bits - 1; idx++)
  91                    {
  92   ✓✗✓✗   388           set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), false);
  93                    }
  94
```

```
                 
                 
          1 | }
                 
                 
          5 | void Driver::set_function_data(bool DSPRPT, bool TMGRST, bool RFRESH, bool ESPWM, bool LSDVLT)
            | {
            | 
            |     // Function
            |     // bits      [===]
            |     //           =][==
            |     // Bytes   #49  #50
            |     enum functions {
            |         tmgrst  = 0,
            |         dsprt   = 1,
            |         lsdvlt  = 5,
            |         espwm   = 6,
            |         rfresh  = 7
            |     };
            | 
            |     // if all are set to true, byte #49 = 3, byte #50 = 224
          5 |     set_value_nth_bit(m_common_byte_register[byte_offsets::function],       functions::dsprt,  DSPRPT);
          5 |     set_value_nth_bit(m_common_byte_register[byte_offsets::function],       functions::tmgrst, TMGRST);
          5 |     set_value_nth_bit(m_common_byte_register[byte_offsets::function + 1],   functions::rfresh, RFRESH);
          5 |     set_value_nth_bit(m_common_byte_register[byte_offsets::function + 1],   functions::espwm,  ESPWM);
          5 |     set_value_nth_bit(m_common_byte_register[byte_offsets::function + 1],   functions::lsdvlt, LSDVLT);
          5 | }
            | 
          2 | void Driver::set_bc_data(
            |     const std::bitset<m_bc_data_resolution> &blue_value,
            |     const std::bitset<m_bc_data_resolution> &green_value,
            |     const std::bitset<m_bc_data_resolution> &red_value)
            | {
            |     // BC         blue   green   red
            |     // bits      [=====][=====][=====]
            |     // bits      ====][======][======]
            |     // Bytes     #50    #51      #52
            | 
            |     // BYTE #50
          2 |     BytePosition byte_pos(byte_offsets::brightness_control);
            | 
            |     // set 5 LSB of byte #50 to bits 6-2 of BC blue_value
     ✓✗   2 |     byte_pos.skip_next_n_bits(3);
 ✓✗✓✗✓✗   2 |     set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), blue_value.test(6));
 ✓✗✓✗✓✗   2 |     set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), blue_value.test(5));
 ✓✗✓✗✓✗   2 |     set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), blue_value.test(4));
 ✓✗✓✗✓✗   2 |     set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), blue_value.test(3));
 ✓✗✓✗✓✗   2 |     set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), blue_value.test(2));
            | 
            |     // BYTE #51
          2 |     byte_pos++;
            | 
            |     // set the 2 MSB bits of byte #51 to the 2 LSB of blue_value
 ✓✗✓✗✓✗   2 |     set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), blue_value.test(1));
 ✓✗✓✗✓✗   2 |     set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), blue_value.test(0));
            | 
            |     // set 5 LSB of byte #51 to bits 6-1 of BC green_value
 ✓✗✓✗✓✗   2 |     set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), green_value.test(6));
 ✓✗✓✗✓✗   2 |     set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), green_value.test(5));
 ✓✗✓✗✓✗   2 |     set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), green_value.test(4));
 ✓✗✓✗✓✗   2 |     set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), green_value.test(3));
 ✓✗✓✗✓✗   2 |     set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), green_value.test(2));
 ✓✗✓✗✓✗   2 |     set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), green_value.test(1));
            | 
            |     // BYTE #52
          2 |     byte_pos++;
            | 
            |     // set MSB of byte #52 to LSB of green_value
 ✓✗✓✗✓✗   2 |     set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), green_value.test(0));
            | 
            |     // set 7 LSB of byte #50 to bits all 7 bits of BC red_value
 ✓✗✓✗✓✗   2 |     set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), red_value.test(6));
 ✓✗✓✗✓✗   2 |     set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), red_value.test(5));
 ✓✗✓✗✓✗   2 |     set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), red_value.test(4));
 ✓✗✓✗✓✗   2 |     set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), red_value.test(3));
 ✓✗✓✗✓✗   2 |     set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), red_value.test(2));
 ✓✗✓✗✓✗   2 |     set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), red_value.test(1));
 ✓✗✓✗✓✗   2 |     set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), red_value.test(0));
            | 
          2 | }
            | 
          2 | void Driver::set_mc_data(
            |     const std::bitset<m_mc_data_resolution> &blue_value,
            |     const std::bitset<m_mc_data_resolution> green_value,
            |     const std::bitset<m_mc_data_resolution> &red_value)
```

```
              {
                  // MC          B  G  R
                  // bits       [=] [=] [=]
                  // bits       [======] [
                  // Bytes        #53    #54

                  // BYTE #53
           2      BytePosition byte_pos(byte_offsets::max_current);

                  // 3 bits of blue in 3 MSB of byte #51 == 128
    ✓✗✓✗   2      set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), blue_value.test(2));
      ✓✗
    ✓✗✓✗   2      set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), blue_value.test(1));
      ✓✗
    ✓✗✓✗   2      set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), blue_value.test(0));
      ✓✗

                  // 3 bits of green in next 3 bits of byte #51 == 144
    ✓✗✓✗   2      set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), green_value.test(2));
      ✓✗
    ✓✗✓✗   2      set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), green_value.test(1));
      ✓✗
    ✓✗✓✗   2      set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), green_value.test(0));
      ✓✗

                  // 2 bits of red in 2 LSB of byte #51 (== 146)
    ✓✗✓✗   2      set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), red_value.test(2));
      ✓✗
    ✓✗✓✗   2      set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), red_value.test(1));
      ✓✗

                  // BYTE #54
           2      byte_pos++;

                  // and 1bit in MSB of byte #52 (== 0)
    ✓✗✓✗   2      set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), red_value.test(0));
      ✓✗

           2  }

          33  bool Driver::set_dc_data(
                  const uint8_t led_idx,
                  const std::bitset<m_dc_data_resolution> &blue_value,
                  const std::bitset<m_dc_data_resolution> &green_value,
                  const std::bitset<m_dc_data_resolution> &red_value)
              {

                  // The switch cases are arranged in descending byte order: 15 -> 0.
                  // Because the tlc5955 common register overlaps byte boundaries of the buffer all loops are unrolled.
                  // This looks a bit nuts but it makes it easier to read and debug rather than a series of disjointed loops.

                  // Common Register-to-Byte array mapping for DC (dot correction) data

                  // ROW #1
                  // DC         B15     G15     R15     B14     G14     R14     B13     G13     R13     B12     G12     R12
                  // bits     [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====]
                  // Bytes    ======] [======] [======] [======] [======] [======] [======] [======] [======] [======] [====
                  //           #54      #55      #56      #57      #58      #59     #60      #61      #62      #63      #64

                  // ROW #2
                  // DC         B11     G11     R11     B10     G10     R10     B9      G9      R9      B8      G8      R8
                  // bits     [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====]
                  // Bytes    ==] [======] [======] [======] [======] [======] [======] [======] [======] [======] [======] [
                  //         #64    #65      #66     #67      #68      #69      #70      #71      #72      #73      #74

                  // ROW #3
                  // DC         B7      G7      R7      B6      G6      R6      B5      G5      R5      B4      G4      R4
                  // bits     [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====]
                  // Bytes    ======] [======] [======] [======] [======] [======] [======] [======] [======] [======] [====
                  //           #75      #76      #77      #78      #79      #80      #81      #82      #83      #84      #85

                  // ROW #4
                  // DC         B3      G3      R3      B2      G2      R2      B1      G1      R1      B0      G0      R0
                  // bits     [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====]
                  // Bytes    ==] [======] [======] [======] [======] [======] [======] [======] [======] [======] [======] [
                  //         #85    #86      #87      #88     #89      #90      #91      #92      #93      #94      #95    #96

          33      BytePosition byte_pos(byte_offsets::dot_correct);

    ✓✓✓✓
    ✓✓✓✓
    ✓✓✓✓  33      switch(led_idx)
    ✓✓✓✓
       ✓
                  {
           2          case 15:    // LED #15

                          // ROW #1
                          // DC         B15     G15     R15
                          // bits     [=====] [=====] [=====]
                          // Bytes    ======] [======] [======]
                          //           #54      #55      #56

                          // LED B15
           2              byte_pos.set_byte_idx(byte_offsets::dot_correct); // BYTE #54
      ✓✗   2              byte_pos.next_bit_idx();    // skip MSB
           2              break;

           2          case 14:    // LED #14

                          // ROW #1
                          // DC         B14     G14     R14
                          // bits     [=====] [=====] [=====]
                          // Bytes    =] [======] [======] [==
                          //        #56    #57      #58     #59

```

```
                    // LED B14
      2             byte_pos.set_byte_idx(byte_offsets::dot_correct + 2); // BYTE #56
 ✓✗   2             byte_pos.skip_next_n_bits(6);
      2             break;

      2         case 13:    // LED #13

                    // ROW #1
                    // DC         B15     G15     R15     B14     G14     R14     B13     G13     R13     B12     G12     R12
                    // bits      [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====]
                    // Bytes    ======] [======] [======] [======] [======] [======] [======] [======] [======] [======] [====
                    //           #54      #55      #56      #57       #58      #59     #60      #61      #62      #63      #64

                    // LED B13
      2             byte_pos.set_byte_idx(byte_offsets::dot_correct + 5); // BYTE #59
 ✓✗   2             byte_pos.skip_next_n_bits(3);
      2             break;

      2         case 12:    // LED #12

                    // ROW #1
                    // DC         B15     G15     R15     B14     G14     R14     B13     G13     R13     B12     G12     R12
                    // bits      [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====]
                    // Bytes    ======] [======] [======] [======] [======] [======] [======] [======] [======] [======] [====
                    //           #54      #55      #56      #57       #58      #59     #60      #61      #62      #63      #64

                    // LED B12
      2             byte_pos.set_byte_idx(byte_offsets::dot_correct + 8); // BYTE #62
      2             break;

      2         case 11:    // LED #11

                    // ROW #2
                    // DC         B11     G11     R11     B10     G10     R10     B9      G9      R9      B8      G8      R8
                    // bits      [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====]
                    // Bytes    ==] [======] [======] [======] [======] [======] [======] [======] [======] [======] [======] [
                    //         #64   #65       #66     #67      #68      #69      #70      #71      #72      #73      #74

                    // LED B11
      2             byte_pos.set_byte_idx(byte_offsets::dot_correct + 10);  // BYTE #64
 ✓✗   2             byte_pos.skip_next_n_bits(5);
      2             break;

      2         case 10:    // LED #10

                    // ROW #2
                    // DC         B11     G11     R11     B10     G10     R10     B9      G9      R9      B8      G8      R8
                    // bits      [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====]
                    // Bytes    ==] [======] [======] [======] [======] [======] [======] [======] [======] [======] [======] [
                    //         #64   #65       #66     #67      #68      #69      #70      #71      #72      #73      #74

                    // LED B10
      2             byte_pos.set_byte_idx(byte_offsets::dot_correct + 13);  // BYTE #67
 ✓✗   2             byte_pos.skip_next_n_bits(2);
      2             break;

      2         case 9: // LED #9

                    // ROW #2
                    // DC         B11     G11     R11     B10     G10     R10     B9      G9      R9      B8      G8      R8
                    // bits      [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====]
                    // Bytes    ==] [======] [======] [======] [======] [======] [======] [======] [======] [======] [======] [
                    //         #64   #65       #66     #67      #68      #69      #70      #71      #72      #73      #74

                    // LED B9
      2             byte_pos.set_byte_idx(byte_offsets::dot_correct + 15);  // BYTE #69
 ✓✗   2             byte_pos.skip_next_n_bits(7);
      2             break;

      2         case 8:     // LED #8

                    // ROW #2
                    // DC         B11     G11     R11     B10     G10     R10     B9      G9      R9      B8      G8      R8
                    // bits      [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====]
                    // Bytes    ==] [======] [======] [======] [======] [======] [======] [======] [======] [======] [======] [
                    //         #64   #65       #66     #67      #68      #69      #70      #71      #72      #73      #74

                    // LED B8
      2             byte_pos.set_byte_idx(byte_offsets::dot_correct + 18);  // BYTE #72
 ✓✗   2             byte_pos.skip_next_n_bits(4);
      2             break;

      2         case 7: // LED #7

                    // ROW #3
                    // DC         B7      G7      R7      B6      G6      R6      B5      G5      R5      B4      G4      R4
                    // bits      [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====]
                    // Bytes    ======] [======] [======] [======] [======] [======] [======] [======] [======] [======] [====
                    //           #75      #76      #77      #78      #79      #80      #81      #82      #83      #84      #85

                    // LED B7
      2             byte_pos.set_byte_idx(byte_offsets::dot_correct + 21); // BYTE #75
 ✓✗   2             byte_pos.skip_next_n_bits(1);
      2             break;

      2         case 6:     // LED #6

                    // ROW #3
                    // DC         B7      G7      R7      B6      G6      R6      B5      G5      R5      B4      G4      R4
                    // bits      [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====]
                    // Bytes    ======] [======] [======] [======] [======] [======] [======] [======] [======] [======] [====
                    //           #75      #76      #77      #78      #79      #80      #81      #82      #83      #84      #85

                    // LED B6
      2             byte_pos.set_byte_idx(byte_offsets::dot_correct + 23);  // BYTE #77
 ✓✗   2             byte_pos.skip_next_n_bits(6);
```

```
         2             break;

         2         case 5: // LED #5

                   // ROW #3
                   // DC        B7      G7     R7    B6     G6     R6     B5     G5     R5     B4     G4     R4
                   // bits    [=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====]
                   // Bytes   ======][======][======][======][======][======][======][======][======][======][====
                   //          #75     #76     #77     #78     #79     #80     #81     #82     #83     #84     #85

                   // LED B5
         2         byte_pos.set_byte_idx(byte_offsets::dot_correct + 26);  // BYTE #80
   ✓✗    2         byte_pos.skip_next_n_bits(3);
         2         break;

         2         case 4:

                   // ROW #3
                   // DC        B7      G7     R7    B6     G6     R6     B5     G5     R5     B4     G4     R4
                   // bits    [=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====]
                   // Bytes   ======][======][======][======][======][======][======][======][======][======][====
                   //          #75     #76     #77     #78     #79     #80     #81     #82     #83     #84     #85

                   // LED B4
         2         byte_pos.set_byte_idx(byte_offsets::dot_correct + 29); //BYTE #83
         2         break;

         2         case 3:

                   // ROW #4
                   // DC        B3     G3     R3      B2     G2    R2      B1    G1     R1     B0     G0     R0
                   // bits    [=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====]
                   // Bytes   ==][======][======][======][======][======][======][======][======][======][======][
                   //        #85   #86     #87      #88    #89     #90     #91     #92     #93     #94     #95   #96

                   // LED B3
         2         byte_pos.set_byte_idx(byte_offsets::dot_correct + 31);   // BYTE #85
   ✓✗    2         byte_pos.skip_next_n_bits(5);

         2         break;

         2         case 2:

                   // ROW #4
                   // DC        B3     G3     R3      B2     G2    R2      B1    G1     R1     B0     G0     R0
                   // bits    [=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====]
                   // Bytes   ==][======][======][======][======][======][======][======][======][======][======][
                   //        #85   #86     #87      #88    #89     #90     #91     #92     #93     #94     #95   #96

                   // LED B2
         2         byte_pos.set_byte_idx(byte_offsets::dot_correct + 34);  // BYTE #88
   ✓✗    2         byte_pos.skip_next_n_bits(2);
         2         break;

         2         case 1:

                   // ROW #4
                   // DC        B3     G3     R3      B2     G2    R2      B1    G1     R1     B0     G0     R0
                   // bits    [=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====]
                   // Bytes   ==][======][======][======][======][======][======][======][======][======][======][
                   //        #85   #86     #87      #88    #89     #90     #91     #92     #93     #94     #95   #96

                   // LED B1
         2         byte_pos.set_byte_idx(byte_offsets::dot_correct + 36);  // BYTE #90
   ✓✗    2         byte_pos.skip_next_n_bits(7);
         2         break;

         2         case 0:

                   // ROW #4
                   // DC        B3     G3     R3      B2     G2    R2      B1    G1     R1     B0     G0     R0
                   // bits    [=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====]
                   // Bytes   ==][======][======][======][======][======][======][======][======][======][======][
                   //        #85   #86     #87      #88    #89     #90     #91     #92     #93     #94     #95   #96

                   // LED B0
         2         byte_pos.set_byte_idx(byte_offsets::dot_correct + 39);  // BYTE #93
   ✓✗    2         byte_pos.skip_next_n_bits(4);
         2         break;

         1         default:    // led_idx > 15
         1             return false;
               }

               // set the bits
   ✓✓  256     for (int8_t blue_idx = 6; blue_idx > -1; blue_idx--)
               {
 ✓✗✓✗
   ✓✗  224         set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), blue_value.test(blue_idx));
               }

               // LED G15
   ✓✓  256     for (int8_t green_idx = 6; green_idx > -1; green_idx--)
               {
 ✓✗✓✗
   ✓✗  224         set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), green_value.test(green_idx));
               }

               // LED R15
   ✓✓  256     for (int8_t red_idx = 6; red_idx > -1; red_idx--)
               {
 ✓✗✓✗
   ✓✗  224         set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), red_value.test(red_idx));
               }

        32     return true;
```

```
                  }

                  void Driver::set_all_dc_data(
                      const std::bitset<m_dc_data_resolution> &blue_value,
                      const std::bitset<m_dc_data_resolution> &green_value,
                      const std::bitset<m_dc_data_resolution> &red_value)
                  {
                      for (uint8_t led_idx = 0; led_idx < m_num_leds_per_chip; led_idx++)
                      {
                          set_dc_data(led_idx, blue_value, green_value, red_value);
                      }
                  }

             33   bool Driver::set_gs_data(
                      uint8_t led_idx,
                      const std::bitset<m_gs_data_resolution> &blue_value,
                      const std::bitset<m_gs_data_resolution> &green_value,
                      const std::bitset<m_gs_data_resolution> &red_value)
                  {
✓✓           33       if (led_idx >= m_num_leds_per_chip)
                      {
              1           return false;
                      }
                      // offset for the current LED position
             32       const uint16_t led_offset = m_gs_data_one_led_size_bits * led_idx;

                      // the current bit position within the GS section of the common register, starting at the section offset + LED offset
             32       uint16_t gs_common_pos = 1U + led_offset;

                      // check gs_common_pos has left enough bits for one segment of LED GS data
                      // This could happen if the header constants are incorrect
✗✓           32       if (gs_common_pos + m_gs_data_one_led_size_bits > m_common_reg_size_bits)
                      {
                          return false;
                      }

                          // ROW #1
                          // GS            Bn              Gn              Rn
                          // bits    0[==============][==============][==============]
                          // Bytes    [======][======][======][======][======][======][
                          //            #0      #1      #2      #3      #4      #5    #6


                      // set the bits

                      // should always give multiple of 6.
             32       uint16_t begin_byte_idx = gs_common_pos / 8;

             32       BytePosition byte_pos(begin_byte_idx);

                      // byte #0, skip the MSB
✓✗           32       byte_pos.skip_next_n_bits(1);

✓✓          544       for (int8_t blue_idx = 15; blue_idx > -1; blue_idx--)
                      {
✓✗✓✗✓✗      512           set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), blue_value.test(blue_idx));
                      }
✓✓          544       for (int8_t green_idx = 15; green_idx > -1; green_idx--)
                      {
✓✗✓✗✓✗      512           set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), green_value.test(green_idx));
                      }
✓✓          544       for (int8_t red_idx = 15; red_idx > -1; red_idx--)
                      {
✓✗✓✗✓✗      512           set_value_nth_bit(m_common_byte_register[byte_pos.get_byte_idx()], byte_pos.next_bit_idx(), red_value.test(red_idx));
                      }

             32       return true;
                  }

                  void Driver::set_all_gs_data(
                      const std::bitset<m_gs_data_resolution> &blue_value,
                      const std::bitset<m_gs_data_resolution> &green_value,
                      const std::bitset<m_gs_data_resolution> &red_value)
                  {
                      for (uint8_t led_idx = 0; led_idx < m_num_leds_per_chip; led_idx++)
                      {
                          set_gs_data(led_idx, blue_value, green_value, red_value);
                      }
                  }




             71   void Driver::send_data()
                  {
                      // clock the data through and latch

                  #ifdef USE_HAL_DRIVER
                      HAL_GPIO_WritePin(m_gsclk_port, m_gsclk_pin, GPIO_PIN_SET);
                      HAL_StatusTypeDef res = HAL_SPI_Transmit(&m_spi_interface, (uint8_t*)m_common_byte_register.data(), m_common_reg_size_bytes, 0x0000'000F);
                      UNUSED(res);
                      HAL_GPIO_WritePin(m_gsclk_port, m_gsclk_pin, GPIO_PIN_RESET);
                  #endif
             71       toggle_latch();
             71   }

             71   void Driver::toggle_latch()
                  {
                  #ifdef USE_HAL_DRIVER
                      HAL_Delay(m_latch_delay_ms);
                      HAL_GPIO_WritePin(m_lat_port, m_lat_pin, GPIO_PIN_SET);
                      HAL_Delay(m_latch_delay_ms);
                      HAL_GPIO_WritePin(m_lat_port, m_lat_pin, GPIO_PIN_RESET);
                      HAL_Delay(m_latch_delay_ms);
                  #endif
               71 }

               71 void Driver::flush_common_register()
                  {
✓✓   6958     for (auto &byte : m_common_byte_register)
                      {
             6887         byte = 0x00;
                      }
               71     send_data();
               71 }

             2690 void Driver::print_common_bits()
                  {
                  #ifdef USE_RTT
                      SEGGER_RTT_printf(0, "\r\n");
                      for (uint16_t idx = 45; idx < 53; idx++)
                      {
                          SEGGER_RTT_printf(0, "%u ", +m_common_byte_register[idx]);
                      }
                  #endif

             2690 }

                  // void Driver::flush_common_register()
                  // {
                  //     // reset the latch
                  //     HAL_GPIO_WritePin(m_lat_port, m_lat_pin, GPIO_PIN_RESET);

                  //     // clock-in the entire common shift register per daisy-chained chip before pulsing the latch
                  //     for (uint8_t shift_entire_reg = 0; shift_entire_reg < m_num_driver_ics; shift_entire_reg++)
                  //     {
                  //         // write the MSB bit low to signal greyscale data
                  //         HAL_GPIO_WritePin(m_sck_port, m_sck_pin, GPIO_PIN_RESET);
                  //         HAL_GPIO_WritePin(m_mosi_port, m_mosi_pin, GPIO_PIN_RESET);
                  //         HAL_GPIO_WritePin(m_sck_port, m_sck_pin, GPIO_PIN_SET);
                  //         HAL_GPIO_WritePin(m_sck_port, m_sck_pin, GPIO_PIN_RESET);

                  //         // Set all 16-bit colours to 0 greyscale
                  //         uint8_t grayscale_data[2] = {0x00, 0x00};
                  //         for (uint8_t idx = 0; idx < 16; idx++)
                  //         {
                  //             HAL_SPI_Transmit(&m_spi_interface, grayscale_data, 2, HAL_MAX_DELAY);
                  //             HAL_SPI_Transmit(&m_spi_interface, grayscale_data, 2, HAL_MAX_DELAY);
                  //             HAL_SPI_Transmit(&m_spi_interface, grayscale_data, 2, HAL_MAX_DELAY);
                  //         }
                  //     }

                  //     toggle_latch();
                  // }

                  // void Driver::enable_spi()
                  // {
                  //     HAL_GPIO_DeInit(GPIOB, TLC5955_SPI2_MOSI_Pin|TLC5955_SPI2_SCK_Pin);

                  //     m_spi_interface.Instance = SPI2;

                  //     m_spi_interface.Init.Mode = SPI_MODE_MASTER;
                  //     m_spi_interface.Init.Direction = SPI_DIRECTION_1LINE;
                  //     m_spi_interface.Init.DataSize = SPI_DATASIZE_8BIT;
                  //     m_spi_interface.Init.CLKPolarity = SPI_POLARITY_LOW;
                  //     m_spi_interface.Init.CLKPhase = SPI_PHASE_1EDGE;
                  //     m_spi_interface.Init.NSS = SPI_NSS_SOFT;
                  //     m_spi_interface.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_8;
                  //     m_spi_interface.Init.FirstBit = SPI_FIRSTBIT_MSB;
                  //     m_spi_interface.Init.TIMode = SPI_TIMODE_DISABLE;
                  //     m_spi_interface.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
                  //     m_spi_interface.Init.CRCPolynomial = 7;
                  //     m_spi_interface.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
                  //     m_spi_interface.Init.NSSPMode = SPI_NSS_PULSE_DISABLE;

                  //     if (HAL_SPI_Init(&m_spi_interface) != HAL_OK) { Error_Handler(); }

                  //     __HAL_RCC_SPI2_CLK_ENABLE();
                  //     __HAL_RCC_GPIOB_CLK_ENABLE();

                  //     GPIO_InitTypeDef GPIO_InitStruct = {
                  //         TLC5955_SPI2_MOSI_Pin|TLC5955_SPI2_SCK_Pin,
                  //         GPIO_MODE_AF_PP,
                  //         GPIO_PULLDOWN,
                  //         GPIO_SPEED_FREQ_VERY_HIGH,
                  //         GPIO_AF1_SPI2,
                  //     };

                  //     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

                  //     __HAL_SYSCFG_FASTMODEPLUS_ENABLE(SYSCFG_FASTMODEPLUS_PB8);

                  // }

                  // void Driver::disable_spi()
                  // {

                  // }

                  // void Driver::enable_gpio_output_only()
                  // {
                  //     // disable SPI config
                  //     __HAL_RCC_SPI2_CLK_DISABLE();
                  //     HAL_GPIO_DeInit(GPIOB, TLC5955_SPI2_MOSI_Pin|TLC5955_SPI2_SCK_Pin);

                  //     // GPIO Ports Clock Enable
                  //     __HAL_RCC_GPIOB_CLK_ENABLE();

                  //     // Configure GPIO pin Output Level
                  //     HAL_GPIO_WritePin(GPIOB, TLC5955_SPI2_LAT_Pin|TLC5955_SPI2_GSCLK_Pin|TLC5955_SPI2_MOSI_Pin|TLC5955_SPI2_SCK_Pin, GPIO_PIN_RESET);

                  //     // Configure GPIO pins
                  //     GPIO_InitTypeDef GPIO_InitStruct = {
                  //         TLC5955_SPI2_LAT_Pin|TLC5955_SPI2_GSCLK_Pin|TLC5955_SPI2_MOSI_Pin|TLC5955_SPI2_SCK_Pin,
                  //         GPIO_MODE_OUTPUT_PP,
                  //         GPIO_PULLDOWN,
                  //         GPIO_SPEED_FREQ_VERY_HIGH,
                  //         0
                  //     };

                  //     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

                  //     __HAL_SYSCFG_FASTMODEPLUS_ENABLE(SYSCFG_FASTMODEPLUS_PB9);
                  //     __HAL_SYSCFG_FASTMODEPLUS_ENABLE(SYSCFG_FASTMODEPLUS_PB6);
                  //     __HAL_SYSCFG_FASTMODEPLUS_ENABLE(SYSCFG_FASTMODEPLUS_PB7);
                  //     __HAL_SYSCFG_FASTMODEPLUS_ENABLE(SYSCFG_FASTMODEPLUS_PB8);

                  // }

                  } // namespace tlc5955
```

Generated by: GCOVR (Version 4.2)

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
Date:	2021-12-05 16:22:01	Lines:	311	354	87.9 %
Legend:	low: < 75.0 % medium: >= 75.0 % high: >= 90.0 %	Branches:	234	501	46.7 %

File	Lines			Branches	
cpp_ssd1306/inc/font.hpp		100.0 %	8 / 8	100.0 %	2 / 2
cpp_ssd1306/inc/ssd1306.hpp		86.0 %	43 / 50	14.5 %	9 / 62
cpp_ssd1306/src/ssd1306.cpp		100.0 %	65 / 65	57.9 %	44 / 76
cpp_tlc5955/inc/byte_position.hpp		100.0 %	23 / 23	83.3 %	5 / 6
cpp_tlc5955/inc/tlc5955.hpp		100.0 %	1 / 1	- %	0 / 0
cpp_tlc5955/src/tlc5955.cpp		96.1 %	171 / 178	55.6 %	174 / 313
main_app/src/mainapp.cpp		0.0 %	0 / 29	0.0 %	0 / 42

Generated by: GCOVR (Version 4.2)

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
File:	main_app/src/mainapp.cpp	Lines:	0	29	0.0 %
Date:	2021-12-05 16:22:01	Branches:	0	42	0.0 %

Line Branch Exec Source

```
/*
 * mainapp.cpp
 *
 *  Created on: 7 Nov 2021
 *      Author: chris
 */

#include "mainapp.hpp"
#include <ssd1306.hpp>
#include <tlc5955.hpp>
#include <chrono>
#include <thread>

#include <sstream>

#ifdef __cplusplus
 extern "C"
 {
#endif



 void mainapp()
 {

  static ssd1306::Font5x7 font;
  static ssd1306::Display oled;
  oled.init();

  // oled.fill(ssd1306::Colour::Black);
  // oled.set_cursor(2, 0);
  // std::stringstream text("Init LEDS");
  // oled.write_string(text, small_font, ssd1306::Colour::White, 3);
  // oled.update_screen();

  std::bitset<tlc5955::Driver::m_bc_data_resolution> led_bc {127};
  std::bitset<tlc5955::Driver::m_mc_data_resolution> led_mc {4};
  std::bitset<tlc5955::Driver::m_dc_data_resolution> led_dc {127};
  std::bitset<tlc5955::Driver::m_gs_data_resolution> led_gs {32767};
  tlc5955::Driver leds;


  leds.set_control_bit(true);
  leds.set_ctrl_cmd_bits();
  leds.set_padding_bits();
  leds.set_function_data(true, true, true, true, true);

  leds.set_bc_data(led_bc, led_bc, led_bc);
  leds.set_mc_data(led_mc, led_mc, led_mc);
  leds.set_all_dc_data(led_dc, led_dc, led_dc);
  leds.send_data();
  leds.flush_common_register();

  //leds.send_control_data();
  uint8_t count = 0;
  uint32_t delay_ms {0};
  while(true)
  {

   std::stringstream msg;
   msg << font.character_map[count];
   oled.write(msg, font, 2, 2, ssd1306::Colour::Black, ssd1306::Colour::White, 3, true);
   if (count < font.character_map.size() - 1) { count++; }
   else { count = 0; }

   leds.set_control_bit(false);
   leds.set_all_gs_data(led_gs, led_gs, led_gs);
   leds.send_data();

   leds.flush_common_register();
#ifdef USE_HAL_DRIVER
   HAL_Delay(delay_ms);
#else
   std::this_thread::sleep_for(std::chrono::milliseconds(delay_ms));
#endif
   // leds.flush_common_register();
   //HAL_Delay(1);
   //HAL_GPIO_WritePin(TLC5955_SPI2_LAT_GPIO_Port, TLC5955_SPI2_LAT_Pin, GPIO_PIN_RESET);
  }
 }


#ifdef __cplusplus
 }
#endif
```

Generated by: GCOVR (Version 4.2)