GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
Date:	2021-11-30 04:05:59	**Lines:**	650	687	94.6 %
Legend:	low: < 75.0 % medium: >= 75.0 % high: >= 90.0 %	**Branches:**	117	249	47.0 %

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_ssd1306/tests/ssd1306_tester.cpp		100.0 %	21 / 21	52.5 %	21 / 40
cpp_ssd1306/tests/ssd1306_tester.hpp		100.0 %	4 / 4	- %	0 / 0
cpp_tlc5955/inc/tlc5955.hpp		100.0 %	1 / 1	- %	0 / 0
cpp_tlc5955/src/tlc5955.cpp		98.6 %	500 / 507	88.9 %	40 / 45
cpp_tlc5955/tests/tlc5955_tester.cpp		38.1 %	8 / 21	10.0 %	1 / 10
main_app/src/mainapp.cpp		0.0 %	0 / 10	0.0 %	0 / 14

GCC Code Coverage Report

		Exec	Total	Coverage
Directory: ./				
File: cpp_ssd1306/inc/font.hpp	Lines:	8	8	100.0 %
Date: 2021-11-30 04:05:59	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		1044	bool get_pixel(size_t idx, uint32_t &bit_line)
29			{
30	✓✓	1044	if (idx > data.size())
31			{
32		4	return false;
33			}
34			else
35			{
36		1040	bit_line = static_cast<uint32_t>(data.at(idx));
37		1040	return true;
38			}
39			}
40			
41			// @brief get the width member variable
42			// @return uint8_t the width value
43		17738	uint8_t width() { return m_width; }
44			
45			// @brief get tte height member variable
46			// @return uint8_t the height value
47		3422	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		20	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			

```
68          // @brief The width of the font in pixels
69          static uint8_t m_width;
70
71          // @brief The height of the font in pixels
72          static uint8_t m_height;
73
74          // @brief the font data
75          static std::array<uint16_t, FONT_SIZE> data;
76
77      };
78
79      // specializations
80      typedef Font<475> Font5x5;
81      typedef Font<680> Font5x7;
82      typedef Font<950> Font7x10;
83      typedef Font<1710> Font11x18;
84      typedef Font<2470> Font16x26;
85
86      } // namespace ssd1306
87
88      #endif // __FONT_HPP__
```

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
File:	cpp_ssd1306/inc/ssd1306.hpp	Lines:	43	50	86.0 %
Date:	2021-11-30 04:05:59	Branches:	9	62	14.5 %

```
Line Branch 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          10     Display() = default;
  45
  46          20     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
```

```
95                  // @param y
96                  bool set_cursor(uint8_t x, uint8_t y);
97
98
99                  // @brief
100                 // @param cmd_byte
101                 bool write_command(uint8_t cmd_byte);
102
103                 // @brief
104                 // @param data_buffer
105                 // @param data_buffer_size
106                 bool write_data(uint8_t* data_buffer, size_t data_buffer_size);
107
108                 // @brief
109                    uint16_t m_currentx {0};
110
111                 // @brief
112                    uint16_t m_currenty {0};
113
114                 // @brief
115                    uint8_t m_inverted {0};
116
117                 // @brief
118                    uint8_t m_initialized {0};
119
120                 // @brief The display width in bytes. Used in std::array.
121                    static const uint16_t m_width {128};
122
123                 // @brief The display height, in bytes. Used in std::array.
124                    static const uint16_t m_height {64};
125
126                 #ifdef USE_HAL_DRIVER
127
128                 // @brief
129                 SPI_HandleTypeDef m_spi_port {hspi1};
130                 // @brief
131                 uint16_t m_cs_port {0};
132                 // @brief
133                 uint16_t m_cs_pin {0};
134                 // @brief
135                 GPIO_TypeDef* m_dc_port {SPI1_DC_GPIO_Port};
136                 // @brief
137                 uint16_t m_dc_pin {SPI1_DC_Pin};
138                 // @brief
139                 GPIO_TypeDef* m_reset_port {SPI1_RESET_GPIO_Port};
140                 // @brief
141                 uint16_t m_reset_pin {SPI1_RESET_Pin};
142
143                 #endif
144
145                 protected:
146
147                 // @brief byte buffer for ssd1306. Access to derived classes like ssd1306_tester is permitted.
148                    std::array<uint8_t, (m_width*m_height)/8> m_buffer;
149
150                 // @brief
151                 // @tparam FONT_SIZE
152                 // @param ss
153                 // @param font
154                 // @param colour
155                 // @param padding
156                 // @return char
157                 template<std::size_t FONT_SIZE>
158                 char write_string(std::stringstream &ss, Font<FONT_SIZE> &font, Colour colour, bool padding);
159
160                 // @brief
161                 // @tparam FONT_SIZE
162                 // @param ch
163                 // @param font
164                 // @param colour
165                 // @param padding
166                 // @return char
167                 template<std::size_t FONT_SIZE>
168                 char write_char(char ch, Font<FONT_SIZE> &font, Colour colour, bool padding);
169
170
171                 // @brief Get the buffer object. Used for testing only.
172                 // @notes use
173                 // @param buffer
174                 //void get_buffer(std::array<uint8_t, (m_width*m_height)/8> &buffer) { buffer = m_buffer; }
175
176                 };
177
178                 // Out-of-class definitions of member function templates
179
180                 template<std::size_t FONT_SIZE>
181          24     char Display::write(std::stringstream &msg, Font<FONT_SIZE> &font, uint8_t x, uint8_t y, Colour bg, Colour fg, bool padding, bool update)
182                 {
183
184          24         fill(bg);
185    ✓✓    24         if (!set_cursor(x, y))
186                     {
187           8          return 0;
188                     }
189          16         char res = write_string(msg, font, fg, padding);
190    ✓✗    16         if (update)
191                     {
192          16             update_screen();
193                     }
194          16         return res;
195                 }
196
```

```
197                 template<std::size_t FONT_SIZE>
198      60         char Display::write_string(std::stringstream &ss, Font<FONT_SIZE> &font, Colour color, bool padding)
199                 {
200                     // Write until null-byte
201                  char ch;
202 ✓✗✓✗ 60             while (ss.get(ch))
        ✓✓
203                     {
204 ✓✗✗✓  40                 if (write_char(ch, font, color, padding) != ch)
205                         {
206                             // Char could not be written
207                             return ch;
208                         }
209                     }
210
211                     // Everything ok
212      20             return ch;
213                 }
214
215                 template<std::size_t FONT_SIZE>
216      22         char Display::write_char(char ch, Font<FONT_SIZE> &font, Colour colour, bool padding)
217                 {
218
219                     // Check remaining space on current line
220 ✗✗✗✗  44             if (m_width <= (m_currentx + font.height()) ||
221   ✗✗  22                 m_width <= (m_currenty + font.height()))
222                     {
223                         // Not enough space on current line
224                         return 0;
225                     }
226
227                     // add extra leading horizontal space
228    ✗✗  22             if (padding)
229                     {
230    ✗✗ 594              for(size_t n = 0; n < font.height(); n++)
231                       {
232 ✗✗✗✗ 572               if (!draw_pixel(m_currentx, (m_currenty + n), Colour::Black))
233                        {
234                         return false;
235                        }
236                       }
237      22                m_currentx += 1;
238                     }
239
240                     // Use the font to write
241                     uint32_t font_data_word;
242    ✗✗ 542           for(size_t font_height_idx = 0; font_height_idx < font.height(); font_height_idx++)
243                     {
244 ✗✗✗✗ 522             if (!font.get_pixel( (ch - 32) * font.height() + font_height_idx, font_data_word )) { return false; }
245
246                 #ifdef ENABLE_SSD1306_TEST_STDOUT
247                   // separator for the font
248                         std::cout << std::endl;
249                 #endif
250
251   ✗✗8840             for(size_t font_width_idx = 0; font_width_idx < font.width(); font_width_idx++)
252                       {
253   ✗✗8320                 if((font_data_word << font_width_idx) & 0x8000)
254                           {
255  ✗✗✗3220                  switch (colour)
256                         {
257      2234                case Colour::White:
258 ✗✗✗✗ 2234                 if (!draw_pixel(m_currentx + font_width_idx, m_currenty + font_height_idx, Colour::White))
259                          {
260                           return false;
261                          }
262      2234                break;
263
264       986                case Colour::Black:
265 ✗✗✗✗  986                 if (!draw_pixel(m_currentx + font_width_idx, m_currenty + font_height_idx, Colour::Black))
266                          {
267                           return false;
268                          }
269       986                break;
270                         }
271                           }
272                           else
273                           {
274  ✗✗✗5100                  switch (colour)
275                         {
276      2758                case Colour::White:
277 ✗✗✗✗ 2758                 if (!draw_pixel(m_currentx + font_width_idx, m_currenty + font_height_idx, Colour::Black))
278                          {
279                           return false;
280                          }
281      2758                break;
282
283      2342                case Colour::Black:
284 ✗✗✗✗ 2342                 if (!draw_pixel(m_currentx + font_width_idx, m_currenty + font_height_idx, Colour::White))
285                          {
286                           return false;
287                          }
288      2342                break;
289                         }
290                           }
291                       }
292                     }
293
294                     // The current space is now taken
295      20             m_currentx += font.width();
296
297                     // add extra leading horizontal space
```

```
298  xx  20      if (padding)
299              {
300          20      m_currentx += 1;
301              }
302
303                  // Return written char for validation
304          20      return ch;
305              }
306
307
308
309              } // namespace ssd1306
310
311              #endif /* Display_HPP_ */
```

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
File:	cpp_ssd1306/src/ssd1306.cpp	Lines:	65	65	100.0 %
Date:	2021-11-30 04:05:59	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		16	bool Display::init()
19			{
20		16	bool res = true;
21			// Reset Display
22		16	reset();
23			
24			// Wait for the screen to boot
25			#ifdef USE_HAL_DRIVER
26			HAL_Delay(100);
27			#endif
28			// Init Display
29	✗✓	16	if (!write_command(0xAE)) { return false; } //display off
30	✗✓	16	if (!write_command(0x20)) { return false; } //Set Memory Addressing Mode
31	✗✓	16	if (!write_command(0x10)) { return false; } // 00,Horizontal Addressing Mode; 01,Vertical Addressing Mode; 10,Page Addressing Mode (RESET); 11,Invalid
32	✗✓	16	if (!write_command(0xB0)) { return false; } //Set Page Start Address for Page Addressing Mode,0-7
33	✗✓	16	if (!write_command(0xC8)) { return false; } //Set COM Output Scan Direction
34	✗✓	16	if (!write_command(0x00)) { return false; } //---set low column address
35	✗✓	16	if (!write_command(0x10)) { return false; } //---set high column address
36	✗✓	16	if (!write_command(0x40)) { return false; } //--set start line address - CHECK
37	✗✓	16	if (!write_command(0x81)) { return false; } //--set contrast control register - CHECK
38	✗✓	16	if (!write_command(0xFF)) { return false; }
39	✗✓	16	if (!write_command(0xA1)) { return false; } //--set segment re-map 0 to 127 - CHECK
40	✗✓	16	if (!write_command(0xA6)) { return false; } //--set normal color
41	✗✓	16	if (!write_command(0xA8)) { return false; } //--set multiplex ratio(1 to 64) - CHECK
42	✗✓	16	if (!write_command(0x3F)) { return false; } //
43	✗✓	16	if (!write_command(0xA4)) { return false; } //0xa4,Output follows RAM content;0xa5,Output ignores RAM content
44	✗✓	16	if (!write_command(0xD3)) { return false; } //-set display offset - CHECK
45	✗✓	16	if (!write_command(0x00)) { return false; } //-not offset
46	✗✓	16	if (!write_command(0xD5)) { return false; } //--set display clock divide ratio/oscillator frequency
47	✗✓	16	if (!write_command(0xF0)) { return false; } //--set divide ratio
48	✗✓	16	if (!write_command(0xD9)) { return false; } //--set pre-charge period
49	✗✓	16	if (!write_command(0x22)) { return false; } //
50	✗✓	16	if (!write_command(0xDA)) { return false; } //--set com pins hardware configuration - CHECK
51	✗✓	16	if (!write_command(0x12)) { return false; }
52	✗✓	16	if (!write_command(0xDB)) { return false; } //--set vcomh
53	✗✓	16	if (!write_command(0x20)) { return false; } //0x20,0.77xVcc
54	✗✓	16	if (!write_command(0x8D)) { return false; } //--set DC-DC enable
55	✗✓	16	if (!write_command(0x14)) { return false; } //
56	✗✓	16	if (!write_command(0xAF)) { return false; } //--turn on Display panel
57			
58			// Clear screen
59		16	fill(Colour::Black);
60			
61			// Flush buffer to screen
62		16	update_screen();
63			
64			// Set default values for screen object
65		16	m_currentx = 0;
66		16	m_currenty = 0;
67			
68		16	m_initialized = 1;
69			
70		16	return res;
71			}
72			
73			
74		28	void Display::fill(Colour color)
75			{
76	✓✓	28700	for(auto &pixel : m_buffer)
77			{
78	✓✓	28672	pixel = (color == Colour::Black) ? 0x00 : 0xFF;
79			}
80		28	}
81			
82		24	bool Display::update_screen()
83			{
84	✓✓	216	for(uint8_t i = 0; i < 8; i++)
85			{
86	✗✓	192	if (!write_command(0xB0 + i)) { return false; }
87	✗✓	192	if (!write_command(0x00)) { return false; }
88	✗✓	192	if (!write_command(0x10)) { return false; }
89	✗✓	192	if (!write_data(&m_buffer[m_width * i], m_width)) { return false; }
90			}
91		24	return true;
92			}
93			
94		8892	bool Display::draw_pixel(uint8_t x, uint8_t y, Colour color)
95			{
96			// Draw in the right color
97	✓✓	8892	if(color == Colour::White)
98			{
99		4576	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

```
105                 {
106  4316               m_buffer[x + (y / 8) * m_width] &= ~(1 << (y % 8));
107                 #ifdef ENABLE_SSD1306_TEST_STDOUT
108                     std::cout << "_";
109                 #endif
110                 }
111
112  8892           return true;
113             }
114
115    12       bool Display::set_cursor(uint8_t x, uint8_t y)
116             {
117 ✓✓✓✓ 12           if(x >= m_width || y >= m_height)
118                 {
119     4               return false;
120                 }
121                 else
122                 {
123     8               m_currentx = x;
124     8               m_currenty = y;
125                 }
126     8           return true;
127             }
128
129
130    16       void Display::reset()
131             {
132              // CS = High (not selected)
133              //HAL_GPIO_WritePin(Display_CS_Port, Display_CS_Pin, GPIO_PIN_SET);
134
135              // Reset the Display
136             #ifdef USE_HAL_DRIVER
137              HAL_GPIO_WritePin(m_reset_port, m_reset_pin, GPIO_PIN_RESET);
138              HAL_Delay(10);
139              HAL_GPIO_WritePin(m_reset_port, m_reset_pin, GPIO_PIN_SET);
140              HAL_Delay(10);
141             #endif
142    16       }
143
144  1024       bool Display::write_command(uint8_t cmd_byte __attribute__((unused)))
145             {
146             #ifdef USE_HAL_DRIVER
147                 HAL_StatusTypeDef res = HAL_OK;
148              //HAL_GPIO_WritePin(m_cs_port, m_cs_pin, GPIO_PIN_RESET); // select Display
149              HAL_GPIO_WritePin(m_dc_port, m_dc_pin, GPIO_PIN_RESET); // command
150              res = HAL_SPI_Transmit(&m_spi_port, (uint8_t *) &cmd_byte, 1, HAL_MAX_DELAY);
151                 if (res != HAL_OK)
152                 {
153                     return false;
154                 }
155                 return true;
156              //HAL_GPIO_WritePin(m_cs_port, m_cs_pin, GPIO_PIN_SET); // un-select Display
157             #else
158  1024           return true;
159             #endif
160             }
161
162   192       bool Display::write_data(uint8_t* data_buffer __attribute__((unused)), size_t data_buffer_size __attribute__((unused)))
163             {
164             #ifdef USE_HAL_DRIVER
165                 HAL_StatusTypeDef res = HAL_OK;
166              //HAL_GPIO_WritePin(m_cs_port, m_cs_pin, GPIO_PIN_RESET); // select Display
167              HAL_GPIO_WritePin(m_dc_port, m_dc_pin, GPIO_PIN_SET); // data
168              res = HAL_SPI_Transmit(&m_spi_port, data_buffer, data_buffer_size, HAL_MAX_DELAY);
169                 if (res != HAL_OK)
170                 {
171                     return false;
172                 }
173                 return true;
174              //HAL_GPIO_WritePin(m_cs_port, m_cs_pin, GPIO_PIN_SET); // un-select Display
175             #else
176   192           return true;
177             #endif
178
179
180
181             }
182
183             } // namespace ssd1306
```

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
File:	cpp_ssd1306/tests/ssd1306_tester.cpp	**Lines:**	21	21	100.0 %
Date:	2021-11-30 04:05:59	**Branches:**	21	40	52.5 %

```
Line Branch Exec  Source
   1                
   2                #include <ssd1306_tester.hpp>
   3                #include <catch2/catch_all.hpp>
   4                #include <array>
   5                #include <iomanip>
   6                #include <numeric>
   7                
   8                namespace ssd1306
   9                {
  10                
  11     ✓✗      8  ssd1306_tester::ssd1306_tester()
  12                {
     ✓✗✓✗
  13     ✓✗✓✗      8      REQUIRE(init());
     ✓✗✗✗
       ✗✗
  14               8  }
  15                
  16               6  bool ssd1306_tester::validate_buffer(std::vector<uint8_t> &validation_buffer)
  17                {
  18     ✓✓        6      if (validation_buffer.size() != m_buffer.size())
  19                      {
  20     ✓✗✓✗      2          std::cout << "Validation buffer error - expected size: " << m_buffer.size()
     ✓✗✓✗
  21       ✓✗      2              << ", actual size: " << validation_buffer.size() << std::endl;
  22               2          return false;
  23                      }
  24                
  25                      static ssd1306::FontTest font_under_test;
  26                      // set the font character
  27     ✓✗        8      std::stringstream msg;
  28     ✓✗        4      msg << font_under_test.character_map[0];
  29                
  30                      // write the font to the buffer
  31     ✓✗        4      write(msg, font_under_test, 0, 0, ssd1306::Colour::Black, ssd1306::Colour::White, true, true);
  32                
  33                      //get_buffer(m_buffer);
  34                
  35               4      auto valid_buffer_iter = validation_buffer.begin();
  36               4      auto valid_buffer_end = validation_buffer.end();
  37                
  38     ✓✓     2054      for (auto& byte : m_buffer)
  39                      {
  40     ✓✓     2052          if (byte != *valid_buffer_iter)
  41                          {
  42               2              return false;
  43                          }
  44     ✓✗     2050          if (valid_buffer_iter != valid_buffer_end)
  45                          {
  46            2050              valid_buffer_iter++;
  47                          }
  48                      }
  49               2      return true;
  50                }
  51                
  52               2  bool ssd1306_tester::dump_buffer_as_hex()
  53                {
  54                #ifdef ENABLE_SSD1306_TEST_STDOUT
  55                    //get_buffer(m_buffer);
  56                    uint8_t row_count {0};
  57                    uint8_t col_count {0};
  58                
  59                    std::cout << +row_count << ":\t";
  60                    for (auto _byte : m_buffer)
  61                    {
  62                        std::cout << "0x" << std::hex << std::setw(2) << std::setfill('0') << +_byte << ", " << std::flush;
  63                
  64                        if (col_count >= 15)
  65                        {
  66                            col_count = 0;
  67                            row_count ++;
  68                
  69                            std::cout << std::endl << std::dec << (row_count * 16) << ":\t" << std::flush;
  70                        }
```

```
71                  else
72                  {
73                      col_count++;
74                  }
75              }
76          #endif
77      2       return true;
78          }
79
80          } // namespace ssd1306
```

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
File:	cpp_ssd1306/tests/ssd1306_tester.hpp	Lines:	4	4	100.0 %
Date:	2021-11-30 04:05:59	Branches:	0	0	- %

```
Line Branch Exec  Source
   1                #ifndef __SSD1306_TESTER_HPP__
   2                #define __SSD1306_TESTER_HPP__
   3
   4                #include <ssd1306.hpp>
   5                #include <vector>
   6
   7                namespace ssd1306
   8                {
   9
  10                // @brief Single font character = 0xDEADBEEF, use to test the sum validation of the ssd1306 buffer
  11                typedef Font<26> FontTest;
  12
  13                // @brief Tester class inherits protected `ssd1106::Display::get_buffer()` accessor
  14                class ssd1306_tester : public ssd1306::Display
  15                {
  16                public:
  17                    ssd1306_tester();
  18
  19                    // @brief Helper function to provide protected access to ssd1306::Display::write_string()
  20                    // @tparam FONT_SIZE
  21                    // @param ss
  22                    // @param font
  23                    // @param colour
  24                    // @param padding
  25                    // @return char
  26                    template<std::size_t FONT_SIZE>
  27                    char test_write_string(std::stringstream &ss, Font<FONT_SIZE> &font, Colour colour, bool padding);
  28
  29                    // @brief Helper function to provide protected access to ssd1306::Display::write_char()
  30                    // @tparam FONT_SIZE
  31                    // @param ch
  32                    // @param font
  33                    // @param colour
  34                    // @param padding
  35                    // @return char
  36                    template<std::size_t FONT_SIZE>
  37                    char test_write_char(char ch, Font<FONT_SIZE> &font, Colour colour, bool padding);
  38
  39                    // @brief prints the contents of the display buffer. Call write() first or buffer maybe empty.
  40                    // @return always true
  41                    bool dump_buffer_as_hex();
  42
  43                    // @brief validate the ssd1306 mem buffer encoding with known test font data encoding
  44                    // @param validation_buffer The data used to validate the ssd1306 mem buffer. See m_valid_fonttest_buffer_contents.
  45                    // @return true If all bytes match
  46                    // @return false If any bytes don't match
  47                    bool validate_buffer(std::vector<uint8_t> &validation_buffer);
  48
  49                    // @brief The data used to validate the ssd1306 mem buffer
  50                    std::vector<uint8_t> m_valid_fonttest_buffer_contents {
  51                        0x00, 0xff, 0x55, 0xaa, 0xff, 0xff, 0xff, 0xff, 0x00, 0xff, 0xaa, 0xff, 0x00, 0xff, 0xff, 0xaa,
  52                        0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  53                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  54                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  55                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  56                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  57                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  58                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  59                        0x00, 0xff, 0x55, 0xaa, 0xff, 0xff, 0xff, 0xff, 0x00, 0xff, 0xaa, 0xff, 0x00, 0xff, 0xff, 0xaa,
  60                        0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  61                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  62                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  63                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  64                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  65                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  66                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  67                        0x00, 0xff, 0x55, 0xaa, 0xff, 0xff, 0xff, 0xff, 0x00, 0xff, 0xaa, 0xff, 0x00, 0xff, 0xff, 0xaa,
  68                        0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

  69                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  70                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  71                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  72                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  73                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  74                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  75                        0x00, 0x03, 0x01, 0x02, 0x03, 0x03, 0x03, 0x03, 0x00, 0x03, 0x02, 0x03, 0x00, 0x03, 0x03, 0x02,
  76                        0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  77                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  78                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  79                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  80                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  81                        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
```

```
82                  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
83                  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
84                  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
85                  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
86                  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
87                  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
88                  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
89                  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
90                  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
91                  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
92                  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
93                  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
94                  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
95                  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
96                  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
97                  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
98                  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
99                  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
100                 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
101                 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
102                 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
103                 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
104                 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
105                 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
106                 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
107                 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
108                 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
109                 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
110                 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
111                 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
112                 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
113                 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
114                 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
115             };
116
117         };
118
119
120         template<std::size_t FONT_SIZE>
121     2   char ssd1306_tester::test_write_string(std::stringstream &ss, Font<FONT_SIZE> &font, Colour colour, bool padding)
122         {
123     2       return write_string(ss, font, colour, padding);
124         }
125
126         template<std::size_t FONT_SIZE>
127     2   char ssd1306_tester::test_write_char(char ch, Font<FONT_SIZE> &font, Colour colour, bool padding)
128         {
129     2       return write_char(ch, font, colour, padding);
130         }
131
132         } // namespace ssd1306
133
134         #endif // __SSD1306_TESTER_HPP__
```

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
File:	cpp_tlc5955/inc/tlc5955.hpp	Lines:	1	1	100.0 %
Date:	2021-11-30 04:05:59	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  120               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                        std::bitset<m_bc_data_resolution> &blue_value,
  58                        std::bitset<m_bc_data_resolution> &green_value,
  59                        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();
```

```
114
115             // @brief Clears (zeroize) the common register and call send_data()
116             void flush_common_register();
117
118             // @brief toggle the latch pin terminal
119             void toggle_latch();
120
121             // @brief Helper function to print bytes as decimal values to RTT. USE_RTT must be defined.
122             void print_common_bits();
123
124         protected:
125
126             static const uint8_t m_common_reg_size_bytes {97};
127             std::array<uint8_t, m_common_reg_size_bytes> m_common_byte_register{0};
128
129         private:
130
131             uint8_t built_in_test_fail {0};
132
133             // Bits required for correct control reg size
134             static const uint16_t m_common_reg_size_bits {769};
135
136
137              // @brief The number of daisy chained driver chips in the circuit.
138             uint8_t m_num_driver_ics {1};
139
140             // @brief The number of colour channels per LED
141             static const uint8_t m_num_colour_chan {3};
142
143             // @brief The number of LEDs per driver chip
144             static const uint8_t m_num_leds_per_chip {16};
145
146
147
148             // the size of each common register section
149             static const uint8_t m_latch_size_bits {1};                                                                                 // 1U
150             static const uint8_t m_ctrl_cmd_size_bits {8};                                                                              // 8U
151             static constexpr uint16_t m_gs_data_one_led_size_bits {m_gs_data_resolution * m_num_colour_chan};                           // 48U
152             static constexpr uint16_t m_gs_data_section_size_bits {m_gs_data_resolution * m_num_leds_per_chip * m_num_colour_chan}; // 768U
153             static const uint8_t m_func_data_section_size_bits {5};                                                                     // 5U
154             static constexpr uint8_t m_bc_data_section_size_bits {m_bc_data_resolution * m_num_colour_chan};                            // 21U
155             static constexpr uint8_t m_mc_data_section_size_bits {m_mc_data_resolution * m_num_colour_chan};                            // 9U
156             static constexpr uint8_t m_dc_data_one_led_size_bits {m_dc_data_resolution * m_num_colour_chan};                            // 21U
157             static constexpr uint16_t m_dc_data_section_size_bits {m_dc_data_resolution * m_num_leds_per_chip * m_num_colour_chan}; // 336U
158             static constexpr uint16_t m_padding_section_size_bits {                                                                     // 389U
159                 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
160             };
161
162             // the offset of each common register section
163             static const uint8_t m_latch_offset {0};
164             static constexpr uint8_t m_ctrl_cmd_offset {static_cast<uint8_t>(m_latch_offset + m_latch_size_bits)};                   // 1U
165             static constexpr uint8_t m_gs_data_offset {static_cast<uint8_t>(m_ctrl_cmd_offset)};                                     // 9U - used in gs data latch only
166             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
167             static constexpr uint16_t m_func_data_offset {static_cast<uint16_t>(m_padding_offset + m_padding_section_size_bits)};   // 9U
168             static constexpr uint16_t m_bc_data_offset {static_cast<uint16_t>(m_func_data_offset + m_func_data_section_size_bits)}; // 398U
169             static constexpr uint16_t m_mc_data_offset {static_cast<uint16_t>(m_bc_data_offset + m_bc_data_section_size_bits)};     // 424U
170             static constexpr uint16_t m_dc_data_offset {static_cast<uint16_t>(m_mc_data_offset + m_mc_data_section_size_bits)};     // 433U
171
172             // @brief Helper function to set/clear one bit of one byte in the common register byte array
173             // @param target The targetted byte in the common register
174             // @param target_idx The bit within that byte to be set/cleared
175             // @param value The boolean value to set at the bit target_idx
176             void set_value_nth_bit(uint8_t &target, uint16_t target_idx, bool value);
177
178
179             std::bitset<m_common_reg_size_bits> m_common_bit_register{0};
180
181             const uint8_t  m_latch_delay_ms {1};
182
183             // @brief Predefined write command.
184             // section 8.3.2.3 "Control Data Latch" (page 21).
185             // section 8.3.2.2 "Grayscale (GS) Data Latch" (page 20).
186             // https://www.ti.com/lit/ds/symlink/tlc5955.pdf
187             std::bitset<8> m_ctrl_cmd {0x96};
188
189             // @brief Predefined flush command
190             std::bitset<8> m_flush_cmd {0};
191
192             // void enable_spi();
193             // void disable_spi();
194
195             // void enable_gpio_output_only();
196         #ifdef USE_HAL_DRIVER
197             // @brief The HAL SPI interface
198             SPI_HandleTypeDef m_spi_interface {hspi2};
199             // @brief Latch GPIO pin
200          uint16_t m_lat_pin {TLC5955_SPI2_LAT_Pin};
201             // @brief Latch terminal GPIO port
202             GPIO_TypeDef* m_lat_port {TLC5955_SPI2_LAT_GPIO_Port};
203             // @brief GreyScale clock GPIO pin
204             uint16_t m_gsclk_pin {TLC5955_SPI2_GSCLK_Pin};
205             // @brief GreyScale clock GPIO port
206             GPIO_TypeDef* m_gsclk_port {TLC5955_SPI2_GSCLK_GPIO_Port};
207             // @brief SPI MOSI GPIO pin
208             uint16_t m_mosi_pin {TLC5955_SPI2_MOSI_Pin};
209             // @brief SPI MOSI GPIO port
210             GPIO_TypeDef* m_mosi_port {TLC5955_SPI2_MOSI_GPIO_Port};
211             // @brief SPI Clock GPIO pin
212             uint16_t m_sck_pin {TLC5955_SPI2_SCK_Pin};
213             // @brief SPI Clock GPIO port
214             GPIO_TypeDef* m_sck_port {TLC5955_SPI2_SCK_GPIO_Port};
215         #endif
216
217         };
218
219         } // tlc5955
```

GCC Code Coverage Report

		Exec	Total	Coverage
Directory:	./			
File:	cpp_tlc5955/src/tlc5955.cpp	Lines:	500	507	98.6 %
Date:	2021-11-30 04:05:59	Branches:	40	45	88.9 %

```
Line Branch  Exec  Source
   1                
   2                #include "tlc5955.hpp"
   3                #include <sstream>
   4                #include <cmath>
   5                #include <cstring>
   6                #ifdef USE_RTT
   7                    #include <SEGGER_RTT.h>
   8                #endif
   9                namespace tlc5955
  10                {
  11                
  12                
  13            12298  void Driver::set_value_nth_bit(uint8_t &target, uint16_t target_idx, bool value)
  14                {
  15          ✓✓12298      if (value) { target |= (1U << target_idx); }
  16             8394      else { target &= ~(1U << target_idx); }
  17            12298      print_common_bits();
  18            12298  }
  19                
  20                
  21                2  void Driver::set_control_bit(bool ctrl_latch)
  22                {
  23                    // Latch
  24                    // bits      =
  25                    // Bytes     [
  26                    //          #0
  27                
  28                    //m_common_bit_register.set(m_latch_offset, ctrl_latch);
  29                2      set_value_nth_bit(m_common_byte_register[0], 7, ctrl_latch);
  30                2  }
  31                
  32                2  void Driver::set_ctrl_cmd_bits()
  33                {
  34                
  35                    // Ctrl      10010110
  36                    // bits      [======]
  37                    // Bytes     ======][
  38                    //             #0    #1
  39                
  40                    // 7 MSB bits of ctrl byte into 7 LSB of byte #0
  41          ✓✓   16      for (int8_t idx = m_ctrl_cmd_size_bits - 1; idx > 0; idx--)
  42                    {
  43               14          set_value_nth_bit(m_common_byte_register[0], idx -1 , m_ctrl_cmd.test(idx));
  44                
  45                    }
  46                
  47                    // the last m_ctrl_cmd bit in to MSB of byte #1
  48                2      set_value_nth_bit(m_common_byte_register[1], 7, m_ctrl_cmd.test(0));
  49                
  50                2  }
  51                
  52                2  void Driver::set_padding_bits()
  53                {
  54                
  55                    // Padding  0  ================================================================================ 79
  56                    // Bytes       ======][======][======][======][======][======][======][======][======][======][
  57                    //               #1      #2       #3     #4       #5      #6       #7      #8       #9     #10
  58                
  59                    // Padding 80  ================================================================================ 159
  60                    // Bytes       ======][======][======][======][======][======][======][======][======][======][
  61                    //              #11     #12      #13     #14      #15     #16      #17     #18      #19     #20
  62                
  63                    // Padding 160 ================================================================================ 239
  64                    // Bytes       ======][======][======][======][======][======][======][======][======][======][
  65                    //              #21     #22      #23     #24      #25     #26      #27     #28      #29     #30
  66                
  67                    // Padding 240 ================================================================================ 319
  68                    // Bytes       ======][======][======][======][======][======][======][======][======][======][
  69                    //              #31     #32      #33     #34      #35     #36      #37     #38      #39     #40
  70                
  71                    // Padding 320 ======================================================================== 389
  72                    // Bytes       ======][======][======][======][======][======][======][======][======][=====
  73                    //              #41     #42      #43     #44      #45     #46      #47     #48      #49
  74                
  75                    // first, we write 7 LSB bits of m_common_byte_register[1] = 0
  76          ✓✓   16      for (int8_t idx = 6; idx > -1; idx--)
  77                    {
  78               14          set_value_nth_bit(m_common_byte_register[1], idx, false);
  79                    }
  80                
  81                    // The next 47 bytes are don't care padding = 0
  82                2      const uint16_t padding_bytes_remaining = 47U;
  83          ✓✓   92      for (uint16_t byte_idx  = 2; byte_idx < padding_bytes_remaining; byte_idx++)
  84                    {
  85          ✓✓  810          for (int8_t bit_idx = 7; bit_idx > -1; bit_idx--)
  86                            {
  87              720                set_value_nth_bit(m_common_byte_register[byte_idx], bit_idx, false);
  88                            }
  89                    }
  90                
  91                    // lastly, we write 6 MSB bits of m_common_byte_register[49] = 0
  92          ✓✓   14      for (int8_t idx = 7; idx > 1; idx--)
  93                    {
  94               12          set_value_nth_bit(m_common_byte_register[49], idx, false);
  95                    }
  96                
  97                2  }
  98                
  99                
 100               10  void Driver::set_function_data(bool DSPRPT, bool TMGRST, bool RFRESH, bool ESPWM, bool LSDVLT)
 101                {
 102                
 103                    // Function
 104                    // bits      [===]
 105                    //            =][==
 106                    // Bytes   #49  #50
 107                
 108                    // if all are set to true, byte #49 = 3, byte #50 = 224
 109               10      set_value_nth_bit(m_common_byte_register[49], 1, DSPRPT);
 110               10      set_value_nth_bit(m_common_byte_register[49], 0, TMGRST);
 111               10      set_value_nth_bit(m_common_byte_register[50], 7, RFRESH);
 112               10      set_value_nth_bit(m_common_byte_register[50], 6, ESPWM);
```

```
113          10      set_value_nth_bit(m_common_byte_register[50], 5, LSDVLT);
114          10  }
115
116          10  void Driver::set_bc_data(
117                      std::bitset<m_bc_data_resolution> &blue_value,
118                      std::bitset<m_bc_data_resolution> &green_value,
119                      std::bitset<m_bc_data_resolution> &red_value)
120                  {
121                      // BC         blue   green   red
122                      // bits      [=====][=====][=====]
123                      // bits      ====][======][======]
124                      // Bytes     #50    #51      #52
125
126                      // set 5 LSB of byte #50 to bits 6-2 of BC blue_value
127    ✓✓    60          for (int8_t bit_idx = m_bc_data_resolution - 1; bit_idx > 1; bit_idx--)
128                      {
129                          // offset the bit position in byte #50 by 2 places.
130          50              set_value_nth_bit(m_common_byte_register[50], bit_idx - 2, blue_value.test(bit_idx));
131                      }
132
133                      // set the first 2 MSB bits of byte #51 to the last 2 LSB of blue_value
134          10          set_value_nth_bit(m_common_byte_register[51], 7, blue_value.test(1));
135          10          set_value_nth_bit(m_common_byte_register[51], 6, blue_value.test(0));
136
137                      // set 5 LSB of byte #51 to bits 6-1 of BC green_value
138    ✓✓    70          for (int8_t bit_idx = m_bc_data_resolution - 1; bit_idx > 0; bit_idx--)
139                      {
140                          // offset the bit position in byte #51 by 1 places.
141          60              set_value_nth_bit(m_common_byte_register[51], bit_idx - 1, green_value.test(bit_idx));
142                      }
143
144                      // set MSB of byte#52 to LSB of green_value
145          10          set_value_nth_bit(m_common_byte_register[52], 7, green_value.test(0));
146
147                      // set 7 LSB of byte #50 to bits all 7 bits of BC red_value
148    ✓✓    80          for (int8_t bit_idx = m_bc_data_resolution - 1; bit_idx > -1; bit_idx--)
149                      {
150                          // No offset for bit position in byte #52.
151          70              set_value_nth_bit(m_common_byte_register[52], bit_idx, red_value.test(bit_idx));
152                      }
153
154          10  }
155
156          26  void Driver::set_mc_data(
157                      const std::bitset<m_mc_data_resolution> &blue_value,
158                      const std::bitset<m_mc_data_resolution> green_value,
159                      const std::bitset<m_mc_data_resolution> &red_value)
160                  {
161                      // MC          B  G  R
162                      // bits       [=][=][=]
163                      // bits       [======][
164                      // Bytes        #53    #54
165
166                      // 3 bits of blue in 3 MSB of byte #51 == 128
167          26          set_value_nth_bit(m_common_byte_register[53], 7, blue_value.test(m_mc_data_resolution - 1));
168          26          set_value_nth_bit(m_common_byte_register[53], 6, blue_value.test(m_mc_data_resolution - 2));
169          26          set_value_nth_bit(m_common_byte_register[53], 5, blue_value.test(m_mc_data_resolution - 3));
170
171                      // 3 bits of green in next 3 bits of byte #51 == 144
172          26          set_value_nth_bit(m_common_byte_register[53], 4, green_value.test(m_mc_data_resolution - 1));
173          26          set_value_nth_bit(m_common_byte_register[53], 3, green_value.test(m_mc_data_resolution - 2));
174          26          set_value_nth_bit(m_common_byte_register[53], 2, green_value.test(m_mc_data_resolution - 3));
175
176                      // 3 bits of red in 2 LSB of byte #51 (== 146) and MSB of byte #52 (== 0)
177          26          set_value_nth_bit(m_common_byte_register[53], 1, red_value.test(m_mc_data_resolution - 1));
178          26          set_value_nth_bit(m_common_byte_register[53], 0, red_value.test(m_mc_data_resolution - 2));
179          26          set_value_nth_bit(m_common_byte_register[54], 7, red_value.test(m_mc_data_resolution - 3));
180
181          26  }
182
183         162  bool Driver::set_dc_data(
184                      const uint8_t led_idx,
185                      const std::bitset<m_dc_data_resolution> &blue_value,
186                      const std::bitset<m_dc_data_resolution> &green_value,
187                      const std::bitset<m_dc_data_resolution> &red_value)
188                  {
189
190                      // The switch cases are arranged in descending byte order: 15 -> 0.
191                      // Because the tlc5955 common register overlaps byte boundaries of the buffer all loops are unrolled.
192                      // This looks a bit nuts but it makes it easier to read and debug rather than a series of disjointed loops.
193
194                      // Common Register-to-Byte array mapping for DC (dot correction) data
195
196                      // ROW #1
197                      // DC         B15    G15    R15    B14    G14    R14    B13    G13    R13    B12    G12    R12
198                      // bits     [=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====]
199                      // Bytes    ======][======][======][======][======][======][======][======][======][======][====
200                      //           #54     #55     #56     #57      #58     #59     #60     #61     #62     #63     #64
201
202                      // ROW #2
203                      // DC         B11    G11    R11    B10    G10    R10    B9     G9     R9     B8     G8     R8
204                      // bits     [=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====]
205                      // Bytes    ==][======][======][======][======][======][======][======][======][======][======][
206                      //         #64   #65      #66    #67     #68     #69     #70     #71     #72     #73     #74
207
208                      // ROW #3
209                      // DC         B7      G7     R7     B6     G6     R6     B5     G5     R5     B4     G4     R4
210                      // bits     [=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====]
211                      // Bytes    ======][======][======][======][======][======][======][======][======][======][====
212                      //           #75     #76     #77     #78     #79     #80     #81     #82     #83     #84     #85
213
214                      // ROW #4
215                      // DC         B3     G3     R3      B2     G2     R2      B1     G1     R1     B0     G0     R0
216                      // bits     [=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====]
217                      // Bytes    ==][======][======][======][======][======][======][======][======][======][======][
218                      //         #85   #86     #87      #88    #89     #90     #91     #92     #93     #94     #95   #96
219
220
221         162          uint8_t byte_idx{0};
222
    ✓✓✓✓
    ✓✓✓✓
223 ✓✓✓✓ 162          switch(led_idx)
    ✓✓✓✓
      ✓
224                  {
225          10          case 15:    // LED #15
226
227                          // ROW #1
228                          // DC         B15    G15    R15    B14    G14    R14    B13    G13    R13    B12    G12    R12
229                          // bits     [=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====]
230                          // Bytes    ======][======][======][======][======][======][======][======][======][======][====
```

```
231                 //          #54     #55     #56     #57      #58     #59    #60     #61     #62     #63     #64
232
233                     // LED B15
234     10              set_value_nth_bit(m_common_byte_register[byte_idx=54], 6, blue_value.test(6));
235     10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, blue_value.test(5));
236     10              set_value_nth_bit(m_common_byte_register[byte_idx], 4, blue_value.test(4));
237     10              set_value_nth_bit(m_common_byte_register[byte_idx], 3, blue_value.test(3));
238     10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, blue_value.test(2));
239     10              set_value_nth_bit(m_common_byte_register[byte_idx], 1, blue_value.test(1));
240     10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, blue_value.test(0));
241                     // LED G15
242     10              set_value_nth_bit(m_common_byte_register[byte_idx=55], 7, green_value.test(6));
243     10              set_value_nth_bit(m_common_byte_register[byte_idx], 6, green_value.test(5));
244     10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, green_value.test(4));
245     10              set_value_nth_bit(m_common_byte_register[byte_idx], 4, green_value.test(3));
246     10              set_value_nth_bit(m_common_byte_register[byte_idx], 3, green_value.test(2));
247     10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, green_value.test(1));
248     10              set_value_nth_bit(m_common_byte_register[byte_idx], 1, green_value.test(0));
249                     // LED R15
250     10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, red_value.test(6));
251     10              set_value_nth_bit(m_common_byte_register[byte_idx=56], 7, red_value.test(5));
252     10              set_value_nth_bit(m_common_byte_register[byte_idx], 6, red_value.test(4));
253     10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, red_value.test(3));
254     10              set_value_nth_bit(m_common_byte_register[byte_idx], 4, red_value.test(2));
255     10              set_value_nth_bit(m_common_byte_register[byte_idx], 3, red_value.test(1));
256     10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, red_value.test(0));
257
258     10              break;
259
260     10          case 14:    // LED #14
261
262                     // ROW #1
263                     // DC        B15    G15    R15    B14    G14    R14    B13    G13    R13    B12    G12    R12
264                     // bits    [=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====]
265                     // Bytes   [======][======][======][======][======][======][======][======][======][======][====
266                     //          #54     #55     #byte_idx     #57      #58     #59    #60     #61     #62     #63     #64
267
268                     // LED B14
269     10              set_value_nth_bit(m_common_byte_register[byte_idx=56], 1, blue_value.test(6));
270     10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, blue_value.test(5));
271     10              set_value_nth_bit(m_common_byte_register[byte_idx=57], 7, blue_value.test(4));
272     10              set_value_nth_bit(m_common_byte_register[byte_idx], 6, blue_value.test(3));
273     10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, blue_value.test(2));
274     10              set_value_nth_bit(m_common_byte_register[byte_idx], 4, blue_value.test(1));
275     10              set_value_nth_bit(m_common_byte_register[byte_idx], 3, blue_value.test(0));
276                     // LED G14
277     10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, green_value.test(6));
278     10              set_value_nth_bit(m_common_byte_register[byte_idx], 1, green_value.test(5));
279     10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, green_value.test(4));
280     10              set_value_nth_bit(m_common_byte_register[byte_idx=58], 7, green_value.test(3));
281     10              set_value_nth_bit(m_common_byte_register[byte_idx], 6, green_value.test(2));
282     10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, green_value.test(1));
283     10              set_value_nth_bit(m_common_byte_register[byte_idx], 4, green_value.test(0));
284                     // LED R14
285     10              set_value_nth_bit(m_common_byte_register[byte_idx], 3, red_value.test(6));
286     10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, red_value.test(5));
287     10              set_value_nth_bit(m_common_byte_register[byte_idx], 1, red_value.test(4));
288     10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, red_value.test(3));
289     10              set_value_nth_bit(m_common_byte_register[byte_idx=59], 7, red_value.test(2));
290     10              set_value_nth_bit(m_common_byte_register[byte_idx], 6, red_value.test(1));
291     10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, red_value.test(0));
292     10              break;
293
294     10          case 13:    // LED #13
295
296                     // ROW #1
297                     // DC        B15    G15    R15    B14    G14    R14    B13    G13    R13    B12    G12    R12
298                     // bits    [=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====]
299                     // Bytes   [======][======][======][======][======][======][======][======][======][======][====
300                     //          #54     #55     #56     #57      #58     #byte_idx    #60     #61     #62     #63     #64
301
302                     // LED B13
303     10              set_value_nth_bit(m_common_byte_register[byte_idx=59], 4, blue_value.test(6));
304     10              set_value_nth_bit(m_common_byte_register[byte_idx], 3, blue_value.test(5));
305     10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, blue_value.test(4));
306     10              set_value_nth_bit(m_common_byte_register[byte_idx], 1, blue_value.test(3));
307     10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, blue_value.test(2));
308     10              set_value_nth_bit(m_common_byte_register[byte_idx=60], 7, blue_value.test(1));
309     10              set_value_nth_bit(m_common_byte_register[byte_idx], 6, blue_value.test(0));
310                     // LED G13
311     10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, green_value.test(6));
312     10              set_value_nth_bit(m_common_byte_register[byte_idx], 4, green_value.test(5));
313     10              set_value_nth_bit(m_common_byte_register[byte_idx], 3, green_value.test(4));
314     10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, green_value.test(3));
315     10              set_value_nth_bit(m_common_byte_register[byte_idx], 1, green_value.test(2));
316     10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, green_value.test(1));
317     10              set_value_nth_bit(m_common_byte_register[byte_idx=61], 7, green_value.test(0));
318                     // LED R13
319     10              set_value_nth_bit(m_common_byte_register[byte_idx], 6, red_value.test(6));
320     10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, red_value.test(5));
321     10              set_value_nth_bit(m_common_byte_register[byte_idx], 4, red_value.test(4));
322     10              set_value_nth_bit(m_common_byte_register[byte_idx], 3, red_value.test(3));
323     10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, red_value.test(2));
324     10              set_value_nth_bit(m_common_byte_register[byte_idx], 1, red_value.test(1));
325     10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, red_value.test(0));
326     10              break;
327
328     10          case 12:    // LED #12
329
330                     // ROW #1
331                     // DC        B15    G15    R15    B14    G14    R14    B13    G13    R13    B12    G12    R12
332                     // bits    [=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====]
333                     // Bytes   [======][======][======][======][======][======][======][======][======][======][====
334                     //          #54     #55     #56     #57      #58     #59    #60     #61     #62     #63     #64
335
336                     // LED B12
337     10              set_value_nth_bit(m_common_byte_register[byte_idx=62], 7, blue_value.test(6));
338     10              set_value_nth_bit(m_common_byte_register[byte_idx], 6, blue_value.test(5));
339     10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, blue_value.test(4));
340     10              set_value_nth_bit(m_common_byte_register[byte_idx], 4, blue_value.test(3));
341     10              set_value_nth_bit(m_common_byte_register[byte_idx], 3, blue_value.test(2));
342     10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, blue_value.test(1));
343     10              set_value_nth_bit(m_common_byte_register[byte_idx], 1, blue_value.test(0));
344                     // LED G12
345     10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, green_value.test(6));
346     10              set_value_nth_bit(m_common_byte_register[byte_idx=63], 7, green_value.test(5));
347     10              set_value_nth_bit(m_common_byte_register[byte_idx], 6, green_value.test(4));
348     10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, green_value.test(3));
349     10              set_value_nth_bit(m_common_byte_register[byte_idx], 4, green_value.test(2));
350     10              set_value_nth_bit(m_common_byte_register[byte_idx], 3, green_value.test(1));
351     10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, green_value.test(0));
352                     // LED R12
```

```
353  10              set_value_nth_bit(m_common_byte_register[byte_idx], 1, red_value.test(6));
354  10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, red_value.test(5));
355  10              set_value_nth_bit(m_common_byte_register[byte_idx=64], 7, red_value.test(4));
356  10              set_value_nth_bit(m_common_byte_register[byte_idx], 6, red_value.test(3));
357  10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, red_value.test(2));
358  10              set_value_nth_bit(m_common_byte_register[byte_idx], 4, red_value.test(1));
359  10              set_value_nth_bit(m_common_byte_register[byte_idx], 3, red_value.test(0));
360  10              break;
361
362  10          case 11:    // LED #11
363
364                  // ROW #2
365                  // DC         B11    G11    R11    B10    G10    R10    B9     G9     R9     B8     G8     R8
366                  // bits     [=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====]
367                  // Bytes    ==][======][======][======][======][======][======][======][======][======][======][
368                  //         #64    #65      #66    #67     #68     #69     #70     #71     #72     #73     #74
369
370
371                      // LED B11
372  10                  set_value_nth_bit(m_common_byte_register[byte_idx=64], 2, blue_value.test(6));
373  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 1, blue_value.test(5));
374  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 0, blue_value.test(4));
375  10                  set_value_nth_bit(m_common_byte_register[byte_idx=65], 7, blue_value.test(3));
376  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 6, blue_value.test(2));
377  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 5, blue_value.test(1));
378  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 4, blue_value.test(0));
379                      // LED G11
380  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 3, green_value.test(6));
381  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 2, green_value.test(5));
382  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 1, green_value.test(4));
383  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 0, green_value.test(3));
384  10                  set_value_nth_bit(m_common_byte_register[byte_idx=66], 7, green_value.test(2));
385  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 6, green_value.test(1));
386  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 5, green_value.test(0));
387                      // LED R11
388  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 4, red_value.test(6));
389  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 3, red_value.test(5));
390  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 2, red_value.test(4));
391  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 1, red_value.test(3));
392  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 0, red_value.test(2));
393  10                  set_value_nth_bit(m_common_byte_register[byte_idx=67], 7, red_value.test(1));
394  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 6, red_value.test(0));
395
396  10                  break;
397  10          case 10:    // LED #10
398
399                  // ROW #2
400                  // DC         B11    G11    R11    B10    G10    R10    B9     G9     R9     B8     G8     R8
401                  // bits     [=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====]
402                  // Bytes    ==][======][======][======][======][======][======][======][======][======][======][
403                  //         #byte_idx    #65      #66     #byte_idx     #68     #69     #70     #71     #72     #73     #74
404
405
406                      // LED B10
407  10                  set_value_nth_bit(m_common_byte_register[byte_idx=67], 5, blue_value.test(6));
408  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 4, blue_value.test(5));
409  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 3, blue_value.test(4));
410  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 2, blue_value.test(3));
411  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 1, blue_value.test(2));
412  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 0, blue_value.test(1));
413  10                  set_value_nth_bit(m_common_byte_register[byte_idx=68], 7, blue_value.test(0));
414                      // LED G10
415  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 6, green_value.test(6));
416  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 5, green_value.test(5));
417  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 4, green_value.test(4));
418  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 3, green_value.test(3));
419  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 2, green_value.test(2));
420  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 1, green_value.test(1));
421  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 0, green_value.test(0));
422                      // LED R10
423  10                  set_value_nth_bit(m_common_byte_register[byte_idx=69], 7, red_value.test(6));
424  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 6, red_value.test(5));
425  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 5, red_value.test(4));
426  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 4, red_value.test(3));
427  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 3, red_value.test(2));
428  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 2, red_value.test(1));
429  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 1, red_value.test(0));
430  10                  break;
431
432  10          case 9: // LED #9
433
434                  // ROW #2
435                  // DC         B11    G11    R11    B10    G10    R10    B9     G9     R9     B8     G8     R8
436                  // bits     [=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====]
437                  // Bytes    ==][======][======][======][======][======][======][======][======][======][======][
438                  //         #64    #65      #66    #67     #68     #69     #70     #71     #72     #73     #74
439
440
441                      // LED B9
442  10                  set_value_nth_bit(m_common_byte_register[byte_idx=69], 0, blue_value.test(6));
443  10                  set_value_nth_bit(m_common_byte_register[byte_idx=70], 7, blue_value.test(5));
444  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 6, blue_value.test(4));
445  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 5, blue_value.test(3));
446  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 4, blue_value.test(2));
447  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 3, blue_value.test(1));
448  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 2, blue_value.test(0));
449                      // LED G9
450  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 1, green_value.test(6));
451  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 0, green_value.test(5));
452  10                  set_value_nth_bit(m_common_byte_register[byte_idx=71], 7, green_value.test(4));
453  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 6, green_value.test(3));
454  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 5, green_value.test(2));
455  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 4, green_value.test(1));
456  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 3, green_value.test(0));
457                      // LED R9
458  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 2, red_value.test(6));
459  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 1, red_value.test(5));
460  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 0, red_value.test(4));
461  10                  set_value_nth_bit(m_common_byte_register[byte_idx=72], 7, red_value.test(3));
462  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 6, red_value.test(2));
463  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 5, red_value.test(1));
464  10                  set_value_nth_bit(m_common_byte_register[byte_idx], 4, red_value.test(0));
465
466  10                  break;
467
468  10          case 8:     // LED #8
469
470                  // ROW #2
471                  // DC         B11    G11    R11    B10    G10    R10    B9     G9     R9     B8     G8     R8
472                  // bits     [=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====]
473                  // Bytes    ==][======][======][======][======][======][======][======][======][======][======][
474                  //         #64    #65      #66    #67     #68     #69     #70     #71     #byte_idx     #73     #74
```

```
                // LED B8
10              set_value_nth_bit(m_common_byte_register[byte_idx=72], 3, blue_value.test(6));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, blue_value.test(5));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 1, blue_value.test(4));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, blue_value.test(3));
10              set_value_nth_bit(m_common_byte_register[byte_idx=73], 7, blue_value.test(2));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 6, blue_value.test(1));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, blue_value.test(0));
                // LED G8
10              set_value_nth_bit(m_common_byte_register[byte_idx], 4, green_value.test(6));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 3, green_value.test(5));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, green_value.test(4));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 1, green_value.test(3));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, green_value.test(2));
10              set_value_nth_bit(m_common_byte_register[byte_idx=74], 7, green_value.test(1));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 6, green_value.test(0));
                // LED R8
10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, red_value.test(6));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 4, red_value.test(5));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 3, red_value.test(4));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, red_value.test(3));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 1, red_value.test(2));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, red_value.test(1));
10              set_value_nth_bit(m_common_byte_register[byte_idx=75], 7, red_value.test(0));

10              break;

10          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
10              set_value_nth_bit(m_common_byte_register[byte_idx=75], 6, blue_value.test(6));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, blue_value.test(5));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 4, blue_value.test(4));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 3, blue_value.test(3));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, blue_value.test(2));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 1, blue_value.test(1));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, blue_value.test(0));
                // LED G7
10              set_value_nth_bit(m_common_byte_register[byte_idx=76], 7, green_value.test(6));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 6, green_value.test(5));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, green_value.test(4));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 4, green_value.test(3));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 3, green_value.test(2));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, green_value.test(1));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 1, green_value.test(0));
                // LED R7
10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, red_value.test(6));
10              set_value_nth_bit(m_common_byte_register[byte_idx=77], 7, red_value.test(5));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 6, red_value.test(4));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, red_value.test(3));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 4, red_value.test(2));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 3, red_value.test(1));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, red_value.test(0));

10              break;

10          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
10              set_value_nth_bit(m_common_byte_register[byte_idx=77], 1, blue_value.test(6));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, blue_value.test(5));
10              set_value_nth_bit(m_common_byte_register[byte_idx=78],  7, blue_value.test(4));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 6, blue_value.test(3));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, blue_value.test(2));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 4, blue_value.test(1));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 3, blue_value.test(0));
                // LED G6
10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, green_value.test(6));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 1, green_value.test(5));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, green_value.test(4));
10              set_value_nth_bit(m_common_byte_register[byte_idx=79], 7, green_value.test(3));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 6, green_value.test(2));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, green_value.test(1));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 4, green_value.test(0));
                // LED R6
10              set_value_nth_bit(m_common_byte_register[byte_idx], 3, red_value.test(6));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, red_value.test(5));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 1, red_value.test(4));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, red_value.test(3));
10              set_value_nth_bit(m_common_byte_register[byte_idx=80], 7, red_value.test(2));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 6, red_value.test(1));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, red_value.test(0));

10              break;

10          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
10              set_value_nth_bit(m_common_byte_register[byte_idx=80], 4, blue_value.test(6));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 3, blue_value.test(5));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, blue_value.test(4));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 1, blue_value.test(3));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, blue_value.test(2));
10              set_value_nth_bit(m_common_byte_register[byte_idx=81], 7, blue_value.test(1));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 6, blue_value.test(0));
                // LED G5
10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, green_value.test(6));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 4, green_value.test(5));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 3, green_value.test(4));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, green_value.test(3));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 1, green_value.test(2));
10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, green_value.test(1));
```

```
597        10            set_value_nth_bit(m_common_byte_register[byte_idx=82], 7, green_value.test(0));
598                      // LED R5
599        10            set_value_nth_bit(m_common_byte_register[byte_idx], 6, red_value.test(6));
600        10            set_value_nth_bit(m_common_byte_register[byte_idx], 5, red_value.test(5));
601        10            set_value_nth_bit(m_common_byte_register[byte_idx], 4, red_value.test(4));
602        10            set_value_nth_bit(m_common_byte_register[byte_idx], 3, red_value.test(3));
603        10            set_value_nth_bit(m_common_byte_register[byte_idx], 2, red_value.test(2));
604        10            set_value_nth_bit(m_common_byte_register[byte_idx], 1, red_value.test(1));
605        10            set_value_nth_bit(m_common_byte_register[byte_idx], 0, red_value.test(0));
606
607        10            break;
608
609        10        case 4:
610
611                  // ROW #3
612                  // DC         B7      G7     R7     B6     G6     R6     B5     G5     R5     B4     G4     R4
613                  // bits     [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====]
614                  // Bytes    ======] [======] [======] [======] [======] [======] [======] [======] [======] [======] [======] [====
615                  //           #75     #76     #77     #78     #79     #80     #81     #82     #83     #84     #85
616
617                      // LED B4
618        10            set_value_nth_bit(m_common_byte_register[byte_idx=83], 7, blue_value.test(6));
619        10            set_value_nth_bit(m_common_byte_register[byte_idx], 6, blue_value.test(5));
620        10            set_value_nth_bit(m_common_byte_register[byte_idx], 5, blue_value.test(4));
621        10            set_value_nth_bit(m_common_byte_register[byte_idx], 4, blue_value.test(3));
622        10            set_value_nth_bit(m_common_byte_register[byte_idx], 3, blue_value.test(2));
623        10            set_value_nth_bit(m_common_byte_register[byte_idx], 2, blue_value.test(1));
624        10            set_value_nth_bit(m_common_byte_register[byte_idx], 1, blue_value.test(0));
625                      // LED G4
626        10            set_value_nth_bit(m_common_byte_register[byte_idx], 0, green_value.test(6));
627        10            set_value_nth_bit(m_common_byte_register[byte_idx=84], 7, green_value.test(5));
628        10            set_value_nth_bit(m_common_byte_register[byte_idx], 6, green_value.test(4));
629        10            set_value_nth_bit(m_common_byte_register[byte_idx], 5, green_value.test(3));
630        10            set_value_nth_bit(m_common_byte_register[byte_idx], 4, green_value.test(2));
631        10            set_value_nth_bit(m_common_byte_register[byte_idx], 3, green_value.test(1));
632        10            set_value_nth_bit(m_common_byte_register[byte_idx], 2, green_value.test(0));
633                      // LED R4
634        10            set_value_nth_bit(m_common_byte_register[byte_idx], 1, red_value.test(6));
635        10            set_value_nth_bit(m_common_byte_register[byte_idx], 0, red_value.test(5));
636        10            set_value_nth_bit(m_common_byte_register[byte_idx=85], 7, red_value.test(4));
637        10            set_value_nth_bit(m_common_byte_register[byte_idx], 6, red_value.test(3));
638        10            set_value_nth_bit(m_common_byte_register[byte_idx], 5, red_value.test(2));
639        10            set_value_nth_bit(m_common_byte_register[byte_idx], 4, red_value.test(1));
640        10            set_value_nth_bit(m_common_byte_register[byte_idx], 3, red_value.test(0));
641
642        10            break;
643
644        10        case 3:
645
646                  // ROW #4
647                  // DC         B3     G3     R3      B2     G2     R2      B1     G1     R1     B0     G0     R0
648                  // bits     [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====]
649                  // Bytes    ==] [======] [======] [======] [======] [======] [======] [======] [======] [======] [======] [
650                  //         #85    #86     #87      #88    #89     #90     #91     #92     #93     #94     #95    #96
651
652                      // LED B3
653        10            set_value_nth_bit(m_common_byte_register[byte_idx=85], 2, blue_value.test(6));
654        10            set_value_nth_bit(m_common_byte_register[byte_idx], 1, blue_value.test(5));
655        10            set_value_nth_bit(m_common_byte_register[byte_idx], 0, blue_value.test(4));
656        10            set_value_nth_bit(m_common_byte_register[byte_idx=86], 7, blue_value.test(3));
657        10            set_value_nth_bit(m_common_byte_register[byte_idx], 6, blue_value.test(2));
658        10            set_value_nth_bit(m_common_byte_register[byte_idx], 5, blue_value.test(1));
659        10            set_value_nth_bit(m_common_byte_register[byte_idx], 4, blue_value.test(0));
660                      // LED G3
661        10            set_value_nth_bit(m_common_byte_register[byte_idx], 3, green_value.test(6));
662        10            set_value_nth_bit(m_common_byte_register[byte_idx], 2, green_value.test(5));
663        10            set_value_nth_bit(m_common_byte_register[byte_idx], 1, green_value.test(4));
664        10            set_value_nth_bit(m_common_byte_register[byte_idx], 0, green_value.test(3));
665        10            set_value_nth_bit(m_common_byte_register[byte_idx=87], 7, green_value.test(2));
666        10            set_value_nth_bit(m_common_byte_register[byte_idx], 6, green_value.test(1));
667        10            set_value_nth_bit(m_common_byte_register[byte_idx], 5, green_value.test(0));
668                      // LED R3
669        10            set_value_nth_bit(m_common_byte_register[byte_idx], 4, red_value.test(6));
670        10            set_value_nth_bit(m_common_byte_register[byte_idx], 3, red_value.test(5));
671        10            set_value_nth_bit(m_common_byte_register[byte_idx], 2, red_value.test(4));
672        10            set_value_nth_bit(m_common_byte_register[byte_idx], 1, red_value.test(3));
673        10            set_value_nth_bit(m_common_byte_register[byte_idx], 0, red_value.test(2));
674        10            set_value_nth_bit(m_common_byte_register[byte_idx=88], 7, red_value.test(1));
675        10            set_value_nth_bit(m_common_byte_register[byte_idx], 6, red_value.test(0));
676
677
678
679        10            break;
680
681
682        10        case 2:
683
684                  // ROW #4
685                  // DC         B3     G3     R3      B2     G2     R2      B1     G1     R1     B0     G0     R0
686                  // bits     [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====] [=====]
687                  // Bytes    ==] [======] [======] [======] [======] [======] [======] [======] [======] [======] [======] [
688                  //         #85    #86     #87      #88    #89     #90     #91     #92     #93     #94     #95    #96
689
690                      // LED B2
691        10            set_value_nth_bit(m_common_byte_register[byte_idx=88], 5, blue_value.test(6));
692        10            set_value_nth_bit(m_common_byte_register[byte_idx], 4, blue_value.test(5));
693        10            set_value_nth_bit(m_common_byte_register[byte_idx], 3, blue_value.test(4));
694        10            set_value_nth_bit(m_common_byte_register[byte_idx], 2, blue_value.test(3));
695        10            set_value_nth_bit(m_common_byte_register[byte_idx], 1, blue_value.test(2));
696        10            set_value_nth_bit(m_common_byte_register[byte_idx], 0, blue_value.test(1));
697        10            set_value_nth_bit(m_common_byte_register[byte_idx=89], 7, blue_value.test(0));
698                      // LED G2
699        10            set_value_nth_bit(m_common_byte_register[byte_idx], 6, green_value.test(6));
700        10            set_value_nth_bit(m_common_byte_register[byte_idx], 5, green_value.test(5));
701        10            set_value_nth_bit(m_common_byte_register[byte_idx], 4, green_value.test(4));
702        10            set_value_nth_bit(m_common_byte_register[byte_idx], 3, green_value.test(3));
703        10            set_value_nth_bit(m_common_byte_register[byte_idx], 2, green_value.test(2));
704        10            set_value_nth_bit(m_common_byte_register[byte_idx], 1, green_value.test(1));
705        10            set_value_nth_bit(m_common_byte_register[byte_idx], 0, green_value.test(0));
706                      // LED R2
707        10            set_value_nth_bit(m_common_byte_register[byte_idx=90], 7, red_value.test(6));
708        10            set_value_nth_bit(m_common_byte_register[byte_idx], 6, red_value.test(5));
709        10            set_value_nth_bit(m_common_byte_register[byte_idx], 5, red_value.test(4));
710        10            set_value_nth_bit(m_common_byte_register[byte_idx], 4, red_value.test(3));
711        10            set_value_nth_bit(m_common_byte_register[byte_idx], 3, red_value.test(2));
712        10            set_value_nth_bit(m_common_byte_register[byte_idx], 2, red_value.test(1));
713        10            set_value_nth_bit(m_common_byte_register[byte_idx], 1, red_value.test(0));
714
715        10            break;
716
717
718
```

```
719        10          case 1:
720
721                    // ROW #4
722                    // DC         B3     G3     R3      B2     G2     R2      B1    G1     R1     B0     G0     R0
723                    // bits     [=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====]
724                    // Bytes    ==][=======][=======][=======][=======][=======][=======][=======][=======][=======][
725                    //         #85   #86     #87      #88    #89     #90     #91     #92     #93     #94     #95   #96
726
727                        // LED B1
728        10              set_value_nth_bit(m_common_byte_register[byte_idx=90], 0, blue_value.test(6));
729        10              set_value_nth_bit(m_common_byte_register[byte_idx=91], 7, blue_value.test(5));
730        10              set_value_nth_bit(m_common_byte_register[byte_idx], 6, blue_value.test(4));
731        10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, blue_value.test(3));
732        10              set_value_nth_bit(m_common_byte_register[byte_idx], 4, blue_value.test(2));
733        10              set_value_nth_bit(m_common_byte_register[byte_idx], 3, blue_value.test(1));
734        10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, blue_value.test(0));
735                        // LED G1
736        10              set_value_nth_bit(m_common_byte_register[byte_idx], 1, green_value.test(6));
737        10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, green_value.test(5));
738        10              set_value_nth_bit(m_common_byte_register[byte_idx=92], 7, green_value.test(4));
739        10              set_value_nth_bit(m_common_byte_register[byte_idx], 6, green_value.test(3));
740        10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, green_value.test(2));
741        10              set_value_nth_bit(m_common_byte_register[byte_idx], 4, green_value.test(1));
742        10              set_value_nth_bit(m_common_byte_register[byte_idx], 3, green_value.test(0));
743                        // LED R1
744        10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, red_value.test(6));
745        10              set_value_nth_bit(m_common_byte_register[byte_idx], 1, red_value.test(5));
746        10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, red_value.test(4));
747        10              set_value_nth_bit(m_common_byte_register[byte_idx=93], 7, red_value.test(3));
748        10              set_value_nth_bit(m_common_byte_register[byte_idx], 6, red_value.test(2));
749        10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, red_value.test(1));
750        10              set_value_nth_bit(m_common_byte_register[byte_idx], 4, red_value.test(0));
751
752        10              break;
753
754
755
756        10          case 0:
757
758                    // ROW #4
759                    // DC         B3     G3     R3      B2     G2     R2      B1    G1     R1     B0     G0     R0
760                    // bits     [=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====][=====]
761                    // Bytes    ==][=======][=======][=======][=======][=======][=======][=======][=======][=======][
762                    //         #85   #86     #87      #88    #89     #90     #91     #92     #93     #94     #95   #96
763
764                        // LED B0
765        10              set_value_nth_bit(m_common_byte_register[byte_idx=93], 3, blue_value.test(6));
766        10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, blue_value.test(5));
767        10              set_value_nth_bit(m_common_byte_register[byte_idx], 1, blue_value.test(4));
768        10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, blue_value.test(3));
769        10              set_value_nth_bit(m_common_byte_register[byte_idx=94], 7, blue_value.test(2));
770        10              set_value_nth_bit(m_common_byte_register[byte_idx], 6, blue_value.test(1));
771        10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, blue_value.test(0));
772                        // LED G0
773        10              set_value_nth_bit(m_common_byte_register[byte_idx], 4, green_value.test(6));
774        10              set_value_nth_bit(m_common_byte_register[byte_idx], 3, green_value.test(5));
775        10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, green_value.test(4));
776        10              set_value_nth_bit(m_common_byte_register[byte_idx], 1, green_value.test(3));
777        10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, green_value.test(2));
778        10              set_value_nth_bit(m_common_byte_register[byte_idx=95], 7, green_value.test(1));
779        10              set_value_nth_bit(m_common_byte_register[byte_idx], 6, green_value.test(0));
780                        // LED R0
781        10              set_value_nth_bit(m_common_byte_register[byte_idx], 5, red_value.test(6));
782        10              set_value_nth_bit(m_common_byte_register[byte_idx], 4, red_value.test(5));
783        10              set_value_nth_bit(m_common_byte_register[byte_idx], 3, red_value.test(4));
784        10              set_value_nth_bit(m_common_byte_register[byte_idx], 2, red_value.test(3));
785        10              set_value_nth_bit(m_common_byte_register[byte_idx], 1, red_value.test(2));
786        10              set_value_nth_bit(m_common_byte_register[byte_idx], 0, red_value.test(1));
787        10              set_value_nth_bit(m_common_byte_register[byte_idx=96], 7, red_value.test(0));
788
789        10              break;
790
791         2          default:    // led_idx > 15
792         2              return false;
793                    }
794
795       160          return true;
796
797                }
798
799                void Driver::set_all_dc_data(
800                    const std::bitset<m_dc_data_resolution> &blue_value,
801                    const std::bitset<m_dc_data_resolution> &green_value,
802                    const std::bitset<m_dc_data_resolution> &red_value)
803                {
804                    for (uint8_t led_idx = 0; led_idx < m_num_leds_per_chip; led_idx++)
805                    {
806                        set_dc_data(led_idx, blue_value, green_value, red_value);
807                    }
808                }
809
810       162      bool Driver::set_gs_data(
811                    uint8_t led_idx,
812                    const std::bitset<m_gs_data_resolution> &blue_value,
813                    const std::bitset<m_gs_data_resolution> &green_value,
814                    const std::bitset<m_gs_data_resolution> &red_value)
815                {
816    ✓✓  162          if (led_idx >= m_num_leds_per_chip)
817                    {
818         2              return false;
819                    }
820                    // offset for the current LED position
821       160          const uint16_t led_offset = m_gs_data_one_led_size_bits * led_idx;
822
823                    // the current bit position within the GS section of the common register, starting at the section offset + LED offset
824       160          uint16_t gs_common_pos = m_gs_data_offset + led_offset;
825
826                    // check gs_common_pos has left enough bits for one segment of LED GS data
827                    // This could happen if the header constants are incorrect
828    ✗✓  160          if (gs_common_pos + m_gs_data_one_led_size_bits > m_common_reg_size_bits)
829                    {
830                        return false;
831                    }
832
833                        // ROW #1

834                        // GS             B15             G15             R15             B14             G14             R14             B13             G13             R13
835                        // bits     0[===============][===============][===============][===============][===============][===============][===============][===============][===============]
836                        // Bytes    [=======][=======][=======][=======][=======][=======][=======][=======][=======][=======][=======][=======][=======][=======][=======][=======][
837                        //             #0      #1      #2      #3      #4      #5      #6      #7      #8      #9      #10     #11     #12     #13     #14     #15     #16     #17
838
839
```

```
840                 // set the bits
841
842                 // should always give multiple of 6.
843     160         uint16_t begin_byte_idx = gs_common_pos / 8;
844
845
846                 // byte #0, skip the MSB
847     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 6, blue_value.test(15));
848     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 5, blue_value.test(14));
849     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 4, blue_value.test(13));
850     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 3, blue_value.test(12));
851     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 2, blue_value.test(11));
852     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 1, blue_value.test(10));
853     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 0, blue_value.test(9));
854
855                 // byte #1
856     160         begin_byte_idx++;
857     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 7, blue_value.test(8));
858     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 6, blue_value.test(7));
859     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 5, blue_value.test(6));
860     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 4, blue_value.test(5));
861     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 3, blue_value.test(4));
862     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 2, blue_value.test(3));
863     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 1, blue_value.test(2));
864     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 0, blue_value.test(1));
865
866                 // byte #2
867     160         begin_byte_idx++;
868     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 7, blue_value.test(0));
869     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 6, green_value.test(15));
870     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 5, green_value.test(14));
871     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 4, green_value.test(13));
872     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 3, green_value.test(12));
873     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 2, green_value.test(11));
874     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 1, green_value.test(10));
875     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 0, green_value.test(9));
876
877                 // byte #3
878     160         begin_byte_idx++;
879     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 7, green_value.test(8));
880     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 6, green_value.test(7));
881     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 5, green_value.test(6));
882     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 4, green_value.test(5));
883     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 3, green_value.test(4));
884     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 2, green_value.test(3));
885     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 1, green_value.test(2));
886     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 0, green_value.test(1));
887
888                 // byte #4
889     160         begin_byte_idx++;
890     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 7, green_value.test(0));
891     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 6, red_value.test(15));
892     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 5, red_value.test(14));
893     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 4, red_value.test(13));
894     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 3, red_value.test(12));
895     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 2, red_value.test(11));
896     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 1, red_value.test(10));
897     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 0, red_value.test(9));
898
899                 // byte #5
900     160         begin_byte_idx++;
901     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 7, red_value.test(8));
902     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 6, red_value.test(7));
903     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 5, red_value.test(6));
904     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 4, red_value.test(5));
905     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 3, red_value.test(4));
906     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 2, red_value.test(3));
907     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 1, red_value.test(2));
908     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 0, red_value.test(1));
909
910                 // byte #5, write the final bit
911     160         begin_byte_idx++;
912     160         set_value_nth_bit(m_common_byte_register[begin_byte_idx], 7, red_value.test(0));
913
914     160         return true;
915             }
916
917             void Driver::set_all_gs_data(
918                 const std::bitset<m_gs_data_resolution> &blue_value,
919                 const std::bitset<m_gs_data_resolution> &green_value,
920                 const std::bitset<m_gs_data_resolution> &red_value)
921             {
922                 for (uint8_t led_idx = 0; led_idx < m_num_leds_per_chip; led_idx++)
923                 {
924                     set_gs_data(led_idx, blue_value, green_value, red_value);
925                 }
926             }
927
928
929
930     162     void Driver::send_data()
931             {
932                 // clock the data through and latch
933             #ifdef USE_HAL_DRIVER
934                 HAL_StatusTypeDef res = HAL_SPI_Transmit(&m_spi_interface, (uint8_t*)m_common_byte_register.data(), m_common_reg_size_bytes, HAL_MAX_DELAY);
935                 UNUSED(res);
936             #endif
937     162         toggle_latch();
938     162     }
939
940     162     void Driver::toggle_latch()
941             {
942             #ifdef USE_HAL_DRIVER
943                 HAL_Delay(m_latch_delay_ms);
944                 HAL_GPIO_WritePin(m_lat_port, m_lat_pin, GPIO_PIN_SET);
945                 HAL_Delay(m_latch_delay_ms);
946                 HAL_GPIO_WritePin(m_lat_port, m_lat_pin, GPIO_PIN_RESET);
947                 HAL_Delay(m_latch_delay_ms);
948             #endif
949     162     }
950
951     162     void Driver::flush_common_register()
952             {
953 ✓✓15876         for (auto &byte : m_common_byte_register)
954                 {
955   15714             byte = 0x00;
956                 }
957     162         send_data();
958     162     }
959
960   12298     void Driver::print_common_bits()
961             {
```

```
962                 #ifdef USE_RTT
963                     SEGGER_RTT_printf(0, "\r\n");
964                     for (uint16_t idx = 45; idx < 53; idx++)
965                     {
966                         SEGGER_RTT_printf(0, "%u ", +m_common_byte_register[idx]);
967                     }
968                 #endif
969 
970     12298       }
971 
972                 // void Driver::flush_common_register()
973                 // {
974                 //     // reset the latch
975                 //     HAL_GPIO_WritePin(m_lat_port, m_lat_pin, GPIO_PIN_RESET);
976 
977                 //     // clock-in the entire common shift register per daisy-chained chip before pulsing the latch
978                 //     for (uint8_t shift_entire_reg = 0; shift_entire_reg < m_num_driver_ics; shift_entire_reg++)
979                 //     {
980                 //         // write the MSB bit low to signal greyscale data
981                 //         HAL_GPIO_WritePin(m_sck_port, m_sck_pin, GPIO_PIN_RESET);
982                 //         HAL_GPIO_WritePin(m_mosi_port, m_mosi_pin, GPIO_PIN_RESET);
983                 //         HAL_GPIO_WritePin(m_sck_port, m_sck_pin, GPIO_PIN_SET);
984                 //         HAL_GPIO_WritePin(m_sck_port, m_sck_pin, GPIO_PIN_RESET);
985 
986                 //         // Set all 16-bit colours to 0 greyscale
987                 //         uint8_t grayscale_data[2] = {0x00, 0x00};
988                 //         for (uint8_t idx = 0; idx < 16; idx++)
989                 //         {
990                 //             HAL_SPI_Transmit(&m_spi_interface, grayscale_data, 2, HAL_MAX_DELAY);
991                 //             HAL_SPI_Transmit(&m_spi_interface, grayscale_data, 2, HAL_MAX_DELAY);
992                 //             HAL_SPI_Transmit(&m_spi_interface, grayscale_data, 2, HAL_MAX_DELAY);
993                 //         }
994                 //     }
995 
996                 //     toggle_latch();
997                 // }
998 
999                 // void Driver::enable_spi()
1000                // {
1001                //     HAL_GPIO_DeInit(GPIOB, TLC5955_SPI2_MOSI_Pin|TLC5955_SPI2_SCK_Pin);
1002 
1003                //     m_spi_interface.Instance = SPI2;
1004                //     m_spi_interface.Init.Mode = SPI_MODE_MASTER;
1005                //     m_spi_interface.Init.Direction = SPI_DIRECTION_1LINE;
1006                //     m_spi_interface.Init.DataSize = SPI_DATASIZE_8BIT;
1007                //     m_spi_interface.Init.CLKPolarity = SPI_POLARITY_LOW;
1008                //     m_spi_interface.Init.CLKPhase = SPI_PHASE_1EDGE;
1009                //     m_spi_interface.Init.NSS = SPI_NSS_SOFT;
1010                //     m_spi_interface.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_8;
1011                //     m_spi_interface.Init.FirstBit = SPI_FIRSTBIT_MSB;
1012                //     m_spi_interface.Init.TIMode = SPI_TIMODE_DISABLE;
1013                //     m_spi_interface.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
1014                //     m_spi_interface.Init.CRCPolynomial = 7;
1015                //     m_spi_interface.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
1016                //     m_spi_interface.Init.NSSPMode = SPI_NSS_PULSE_DISABLE;
1017 
1018                //     if (HAL_SPI_Init(&m_spi_interface) != HAL_OK) { Error_Handler(); }
1019 
1020                //     __HAL_RCC_SPI2_CLK_ENABLE();
1021                //     __HAL_RCC_GPIOB_CLK_ENABLE();
1022 
1023                //     GPIO_InitTypeDef GPIO_InitStruct = {
1024                //         TLC5955_SPI2_MOSI_Pin|TLC5955_SPI2_SCK_Pin,
1025                //         GPIO_MODE_AF_PP,
1026                //         GPIO_PULLDOWN,
1027                //         GPIO_SPEED_FREQ_VERY_HIGH,
1028                //         GPIO_AF1_SPI2,
1029                //     };
1030 
1031                //     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
1032 
1033                //     __HAL_SYSCFG_FASTMODEPLUS_ENABLE(SYSCFG_FASTMODEPLUS_PB8);
1034 
1035                // }
1036 
1037                // void Driver::disable_spi()
1038                // {
1039 
1040                // }
1041 
1042                // void Driver::enable_gpio_output_only()
1043                // {
1044                //     // disable SPI config
1045                //     __HAL_RCC_SPI2_CLK_DISABLE();
1046                //     HAL_GPIO_DeInit(GPIOB, TLC5955_SPI2_MOSI_Pin|TLC5955_SPI2_SCK_Pin);
1047 
1048                //     // GPIO Ports Clock Enable
1049                //     __HAL_RCC_GPIOB_CLK_ENABLE();
1050 
1051                //     // Configure GPIO pin Output Level
1052                //     HAL_GPIO_WritePin(GPIOB, TLC5955_SPI2_LAT_Pin|TLC5955_SPI2_GSCLK_Pin|TLC5955_SPI2_MOSI_Pin|TLC5955_SPI2_SCK_Pin, GPIO_PIN_RESET);
1053 
1054                //     // Configure GPIO pins
1055                //     GPIO_InitTypeDef GPIO_InitStruct = {
1056                //         TLC5955_SPI2_LAT_Pin|TLC5955_SPI2_GSCLK_Pin|TLC5955_SPI2_MOSI_Pin|TLC5955_SPI2_SCK_Pin,
1057                //         GPIO_MODE_OUTPUT_PP,
1058                //         GPIO_PULLDOWN,
1059                //         GPIO_SPEED_FREQ_VERY_HIGH,
1060                //         0
1061                //     };
1062 
1063                //     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
1064 
1065                //     __HAL_SYSCFG_FASTMODEPLUS_ENABLE(SYSCFG_FASTMODEPLUS_PB9);
1066                //     __HAL_SYSCFG_FASTMODEPLUS_ENABLE(SYSCFG_FASTMODEPLUS_PB6);
1067                //     __HAL_SYSCFG_FASTMODEPLUS_ENABLE(SYSCFG_FASTMODEPLUS_PB7);
1068                //     __HAL_SYSCFG_FASTMODEPLUS_ENABLE(SYSCFG_FASTMODEPLUS_PB8);
1069 
1070                // }
1071 
1072                } // namespace tlc5955
```

GCC Code Coverage Report

		Exec	Total	Coverage
Directory:	./			
File:	cpp_tlc5955/tests/tlc5955_tester.cpp	Lines:	8	21	38.1 %
Date:	2021-11-30 04:05:59	Branches:	1	10	10.0 %

Line	Branch	Exec	Source
1			
2			#include <tlc5955_tester.hpp>
3			
4			#include <iomanip>
5			
6			namespace tlc5955 {
7			
8		1784	bool tlc5955_tester::get_common_reg_at(uint16_t idx, uint8_t &value)
9			{
10	✗✓	1784	if (idx > m_common_reg_size_bytes)
11			{
12			std::cout << "Error at tlc5955_tester::get_common_reg_at() - out of bounds! Max is "
13			<< +m_common_reg_size_bytes << ", received " << idx << std::endl;
14			return false;
15			}
16		1784	value = m_common_byte_register.at(idx);
17		1784	return true;
18			}
19			
20			void tlc5955_tester::print_register(bool dec_format, bool hex_format)
21			{
22			std::cout << std::endl;
23			int count {0};
24			
25			for (auto &byte : m_common_byte_register)
26			{
27			if (count % 8 == 0) { std::cout << std::endl; }
28			
29			if (dec_format) { std::cout << " " << std::dec << std::setw(3) << +byte; }
30			if (hex_format) { std::cout << " 0x" << std::hex << std::setw(2) << std::setfill('0') << +byte; }
31			std::cout << "\t" << std::flush;
32			count++;
33			}
34			std::cout << std::endl;
35			}
36			
37		6	tlc5955_tester::data_t::iterator tlc5955_tester::data_begin()
38			{
39		6	return m_common_byte_register.begin();
40			}
41			
42		6	tlc5955_tester::data_t::iterator tlc5955_tester::data_end()
43			{
44		6	return m_common_byte_register.end();
45			}
46			
47			
48			} // namespace tlc5955

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
Date:	2021-11-30 04:05:59	**Lines:**	650	687	94.6 %
Legend:	low: < 75.0 % medium: >= 75.0 % high: >= 90.0 %	**Branches:**	117	249	47.0 %

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_ssd1306/tests/ssd1306_tester.cpp		100.0 %	21 / 21	52.5 %	21 / 40
cpp_ssd1306/tests/ssd1306_tester.hpp		100.0 %	4 / 4	- %	0 / 0
cpp_tlc5955/inc/tlc5955.hpp		100.0 %	1 / 1	- %	0 / 0
cpp_tlc5955/src/tlc5955.cpp		98.6 %	500 / 507	88.9 %	40 / 45
cpp_tlc5955/tests/tlc5955_tester.cpp		38.1 %	8 / 21	10.0 %	1 / 10
main_app/src/mainapp.cpp		0.0 %	0 / 10	0.0 %	0 / 14

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
File:	main_app/src/mainapp.cpp	Lines:	0	10	0.0 %
Date:	2021-11-30 04:05:59	Branches:	0	14	0.0 %

```
Line Branch  Exec  Source
   1                /*
   2                 * mainapp.cpp
   3                 *
   4                 *  Created on: 7 Nov 2021
   5                 *      Author: chris
   6                 */
   7
   8                #include "mainapp.hpp"
   9                #include <ssd1306.hpp>
  10                #include <tlc5955.hpp>
  11                #include <chrono>
  12                #include <thread>
  13
  14                #include <sstream>
  15
  16                #ifdef __cplusplus
  17                 extern "C"
  18                 {
  19                #endif
  20
  21
  22
  23          #####  void mainapp()
  24                 {
  25
  26                  static ssd1306::Font5x7 font;
  27          #####   static ssd1306::Display oled;
  28          #####   oled.init();
  29
  30                  // oled.fill(ssd1306::Colour::Black);
  31                  // oled.set_cursor(2, 0);
  32                  // std::stringstream text("Init LEDS");
  33                  // oled.write_string(text, small_font, ssd1306::Colour::White, 3);
  34                  // oled.update_screen();
  35
  36                  // std::bitset<tlc5955::Driver::m_bc_data_resolution> led_bc {127};
  37                  // std::bitset<tlc5955::Driver::m_mc_data_resolution> led_mc {4};
  38                  // std::bitset<tlc5955::Driver::m_dc_data_resolution> led_dc {127};
  39                  // std::bitset<tlc5955::Driver::m_gs_data_resolution> led_gs {32767};
  40                  // tlc5955::Driver leds;
  41
  42                  // leds.startup_tests();
  43
  44                  // leds.set_control_bit(true);
  45                  // leds.set_ctrl_cmd_bits();
  46                  // leds.set_padding_bits();
  47                  // leds.set_function_data(true, true, true, true, true);
  48
  49                  // leds.set_bc_data(led_bc, led_bc, led_bc);
  50                  // leds.set_mc_data(led_mc, led_mc, led_mc);
  51                  // // leds.set_all_dc_data(led_dc, led_dc, led_dc);
  52                  // leds.send_data();
  53                  //leds.flush_common_register();
  54
  55                  //leds.send_control_data();
  56          #####   uint8_t count = 0;
  57
  58                  while(true)
  59                  {
  60
  61          #####    std::stringstream msg;
  62          #####    msg << font.character_map[count];
  63          #####    oled.write(msg, font, 2, 2, ssd1306::Colour::Black, ssd1306::Colour::White, 3, true);
  64          #####    if (count < font.character_map.size() - 1) { count++; }
  65          #####    else { count = 0; }
  66
  67                   //leds.set_control_bit(false);
```

```
68          //leds.set_all_gs_data(led_gs, led_gs, led_gs);
69         //  leds.send_data();
70          //leds.flush_common_register();
71        #ifdef USE_HAL_DRIVER
72          HAL_Delay(1000);
73        #else
74          std::this_thread::sleep_for(std::chrono::milliseconds(1000));
75        #endif
76          // leds.flush_common_register();
77          //HAL_Delay(1);
78          //HAL_GPIO_WritePin(TLC5955_SPI2_LAT_GPIO_Port, TLC5955_SPI2_LAT_Pin, GPIO_PIN_RESET);
79         }
80        }
81
82
83        #ifdef __cplusplus
84        }
85        #endif
```