GCC Code Coverage Report

		Exec	Total	Coverage
Directory:	./			
Date:	2021-11-28 15:51:03	**Lines:**	297	445	66.7 %
Legend:	low: < 75.0 % medium: >= 75.0 % high: >= 90.0 %	**Branches:**	179	415	43.1 %

File	Lines			Branches	
cpp_ssd1306/inc/font.hpp		100.0 %	8 / 8	100.0 %	2 / 2
cpp_ssd1306/inc/ssd1306.hpp		86.3 %	44 / 51	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 %	33 / 33	56.8 %	25 / 44
cpp_ssd1306/tests/ssd1306_tester.hpp		100.0 %	4 / 4	- %	0 / 0
cpp_tlc5955/src/tlc5955.cpp		52.2 %	143 / 274	45.6 %	99 / 217
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: 2021-11-28 15:51:03	Branches:	2	2	100.0 %

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

```
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:	44	51	86.3 %
Date:	2021-11-28 15:51:03	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              5  Display() = default;
  45
  46             10  virtual ~Display() = default;
  47
  48                 // @brief
  49                 bool init();
  50
  51
  52                 // @brief
  53                 // @tparam FONT_SIZE
  54                 // @param msg
  55                 // @param font
  56                 // @param x
  57                 // @param y
  58                 // @param bg
  59                 // @param fg
  60                 // @param padding
  61                 // @param update
  62                 // @return char
  63                 template<std::size_t FONT_SIZE>
  64                 char write(std::stringstream &msg, Font<FONT_SIZE> &font, uint8_t x, uint8_t y, Colour bg, Colour fg, bool padding, bool update);
  65
  66
  67                 // @brief Get the display width. Can be used to create a std::array
  68                 // @return constexpr uint16_t
  69                 static constexpr uint16_t get_display_width() { return m_width; }
  70
  71                 // @brief Get the display height. Can be used to create a std::array
  72                 // @return constexpr uint16_t
  73                 static constexpr uint16_t get_display_height() { return m_height; }
  74
  75                private:
  76
  77                 // @brief
  78                 // @param x
  79                 // @param y
  80                 // @param colour
  81                 bool draw_pixel(uint8_t x, uint8_t y, Colour colour);
  82
  83                 // @brief
  84                 // @param colour
  85                 void fill(Colour colour);
  86
  87                 // @brief
  88                 bool update_screen();
  89
  90                 // @brief
  91                 void reset();
  92
  93                 // @brief Set the cursor object
  94                 // @param x
```

```
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                 // @brief byte buffer for ssd1306
127                    std::array<uint8_t, (m_width*m_height)/8> m_buffer;
128
129                 #ifdef USE_HAL_DRIVER
130
131                 // @brief
132                 SPI_HandleTypeDef m_spi_port {hspi1};
133                 // @brief
134                 uint16_t m_cs_port {0};
135                 // @brief
136                 uint16_t m_cs_pin {0};
137                 // @brief
138                 GPIO_TypeDef* m_dc_port {SPI1_DC_GPIO_Port};
139                 // @brief
140                 uint16_t m_dc_pin {SPI1_DC_Pin};
141                 // @brief
142                 GPIO_TypeDef* m_reset_port {SPI1_RESET_GPIO_Port};
143                 // @brief
144                 uint16_t m_reset_pin {SPI1_RESET_Pin};
145
146                 #endif
147
148                 protected:
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       3         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       12        char Display::write(std::stringstream &msg, Font<FONT_SIZE> &font, uint8_t x, uint8_t y, Colour bg, Colour fg, bool padding, bool update)
182                 {
183
184       12            fill(bg);
185    ✓✓ 12            if (!set_cursor(x, y))
186                     {
187       4              return 0;
188                     }
189       8             char res = write_string(msg, font, fg, padding);
190    ✓✗ 8             if (update)
191                     {
192       8                 update_screen();
193                     }
194       8             return res;
195                 }
196
```

```
197                 template<std::size_t FONT_SIZE>
198          30     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  ✓✗✓✗✓✓  30         while (ss.get(ch))
203                     {
204  ✓✗✗✓    20             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          10         return ch;
213                 }
214
215                 template<std::size_t FONT_SIZE>
216          11     char Display::write_char(char ch, Font<FONT_SIZE> &font, Colour colour, bool padding)
217                 {
218
219                     // Check remaining space on current line
220  ✗✗✗✗    22         if (m_width <= (m_currentx + font.height()) ||
221  ✗✗      11             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    ✗✗    11         if (padding)
229                       {
230    ✗✗   297          for(size_t n = 0; n < font.height(); n++)
231                        {
232  ✗✗✗✗   286          if (!draw_pixel(m_currentx, (m_currenty + n), Colour::Black))
233                         {
234                          return false;
235                         }
236                        }
237          11           m_currentx += 1;
238                       }
239
240                     // Use the font to write
241                     uint32_t font_data_word;
242    ✗✗    271         for(size_t font_height_idx = 0; font_height_idx < font.height(); font_height_idx++)
243                      {
244  ✗✗✗✗   261          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    ✗✗   4420          for(size_t font_width_idx = 0; font_width_idx < font.width(); font_width_idx++)
252                        {
253    ✗✗   4160              if((font_data_word << font_width_idx) & 0x8000)
254                            {
255   ✗✗✗   1610               switch (colour)
256                             {
257           1117              case Colour::White:
258  ✗✗✗✗     1117               if (!draw_pixel(m_currentx + font_width_idx, m_currenty + font_height_idx, Colour::White))
259                                 {
260                                  return false;
261                                 }
262           1117                  break;
263
264            493              case Colour::Black:
265  ✗✗✗✗      493               if (!draw_pixel(m_currentx + font_width_idx, m_currenty + font_height_idx, Colour::Black))
266                                 {
267                                  return false;
268                                 }
269            493                  break;
270                             }
271                            }
272                            else
273                            {
274   ✗✗✗   2550                switch (colour)
275                             {
276           1379              case Colour::White:
277  ✗✗✗✗     1379               if (!draw_pixel(m_currentx + font_width_idx, m_currenty + font_height_idx, Colour::Black))
278                                 {
279                                  return false;
280                                 }
281           1379                  break;
282
283           1171              case Colour::Black:
284  ✗✗✗✗     1171               if (!draw_pixel(m_currentx + font_width_idx, m_currenty + font_height_idx, Colour::White))
285                                 {
286                                  return false;
287                                 }
288           1171                  break;
289                             }
290                            }
291                       }
292                     }
293
294                     // The current space is now taken
295          10         m_currentx += font.width();
296
297                     // add extra leading horizontal space
```

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

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
File:	cpp_ssd1306/src/ssd1306.cpp	Lines:	65	65	100.0 %
Date:	2021-11-28 15:51:03	Branches:	44	76	57.9 %

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

```
105                 {
106  2158               m_buffer[x + (y / 8) * m_width] &= ~(1 << (y % 8));
107                 #ifdef ENABLE_SSD1306_TEST_STDOUT
108                     std::cout << "_";
109                 #endif
110                 }
111
112  4446           return true;
113             }
114
115     6       bool Display::set_cursor(uint8_t x, uint8_t y)
116             {
117  ✓✓✓✓   6           if(x >= m_width || y >= m_height)
118                     {
119     2               return false;
120                     }
121                     else
122                     {
123     4               m_currentx = x;
124     4               m_currenty = y;
125                     }
126     4               return true;
127             }
128
129
130     8       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     8       }
143
144   512       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   512           return true;
159             #endif
160             }
161
162    96       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    96           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:**	33	33	100.0 %
Date:	2021-11-28 15:51:03	**Branches:**	25	44	56.8 %

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			
12			
13	✓✗	4	ssd1306_tester::ssd1306_tester()
14			{
15	✓✗✓✗ ✓✗✓✗ ✓✗✗✗ ✗✗	4	REQUIRE(init());
16		4	}
17			
18		3	bool ssd1306_tester::validate_buffer(std::vector<uint8_t> &validation_buffer)
19			{
20	✓✓	3	if (validation_buffer.size() != m_buffer.size())
21			{
22	✓✗✓✗	1	std::cout << "Validation buffer error - expected size: " << m_buffer.size()
23	✓✗✓✗ ✓✗	1	<< ", actual size: " << validation_buffer.size() << std::endl;
24		1	return false;
25			}
26			
27			static ssd1306::FontTest font_under_test;
28			// set the font character
29	✓✗	4	std::stringstream msg;
30	✓✗	2	msg << font_under_test.character_map[0];
31			
32			// write the font to the buffer
33	✓✗	2	write(msg, font_under_test, 0, 0, ssd1306::Colour::Black, ssd1306::Colour::White, true, true);
34			
35		2	get_buffer(m_buffer);
36			
37		2	auto valid_buffer_iter = validation_buffer.begin();
38		2	auto valid_buffer_end = validation_buffer.end();
39			
40	✓✓	1027	for (auto& byte : m_buffer)
41			{
42	✓✓	1026	if (byte != *valid_buffer_iter)
43			{
44		1	return false;
45			}
46	✓✗	1025	if (valid_buffer_iter != valid_buffer_end)
47			{
48		1025	valid_buffer_iter++;
49			}
50			}
51		1	return true;
52			}
53			
54		1	bool ssd1306_tester::dump_buffer_as_hex()
55			{
56		1	get_buffer(m_buffer);
57		1	uint8_t row_count {0};
58		1	uint8_t col_count {0};
59			
60		1	std::cout << +row_count << ":\t";
61	✓✓	1025	for (auto _byte : m_buffer)
62			{
63		1024	std::cout << "0x" << std::hex << std::setw(2) << std::setfill('0') << +_byte << ", " << std::flush;
64			
65	✓✓	1024	if (col_count >= 15)
66			{
67		64	col_count = 0;
68		64	row_count ++;
69			
70		64	std::cout << std::endl << std::dec << (row_count * 16) << ":\t" << std::flush;

```
71                  }
72                  else
73                  {
74      960             col_count++;
75                  }
76              }
77
78        1     return true;
79          }
80
81          } // namespace ssd1306
```

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
File:	cpp_ssd1306/tests/ssd1306_tester.hpp	Lines:	4	4	100.0 %
Date:	2021-11-28 15:51:03	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         private:
118
119             // @brief used to calculate display size
120             static const size_t byte_size {8};
121
122             // @brief calculate the display dimensions at compile time
123             static constexpr size_t display_size = get_display_width() * get_display_height() / byte_size;
124
125             // @brief local copy of the ssd1306::Display data buffer
126             std::array<uint8_t, display_size> m_buffer;
127         };
128
129
130         template<std::size_t FONT_SIZE>
131       1 char ssd1306_tester::test_write_string(std::stringstream &ss, Font<FONT_SIZE> &font, Colour colour, bool padding)
132         {
133       1     return write_string(ss, font, colour, padding);
134         }
135
136         template<std::size_t FONT_SIZE>
137       1 char ssd1306_tester::test_write_char(char ch, Font<FONT_SIZE> &font, Colour colour, bool padding)
138         {
139       1     return write_char(ch, font, colour, padding);
140         }
141
142         } // namespace ssd1306
143
144         #endif // __SSD1306_TESTER_HPP__
```

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	./				
File:	cpp_tlc5955/src/tlc5955.cpp	Lines:	143	274	52.2 %
Date:	2021-11-28 15:51:03	Branches:	99	217	45.6 %

```
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          1     uint16_t Driver::startup_tests()
  13                {
  14                    // latch bit test
  15    ✗✓    1         if (m_common_byte_register[0] != 0b00000000) built_in_test_fail++;
  16    ✓✗    1         set_control_bit(true);
  17    ✗✓    1         if (m_common_byte_register[0] != 0b10000000) built_in_test_fail++;      // 128
  18                
  19                    // control byte test
  20                
  21                    // Ctrl      10010110
  22                    // bits      [======]
  23                    // Bytes     ======][
  24                    //             #0    #1
  25                
  26    ✓✗    1         set_ctrl_cmd_bits();
  27    ✗✓    1         if (m_common_byte_register[0] != 0b11001011) built_in_test_fail++;      // 203
  28                
  29                    // padding bits test - bytes 1-48 should be empty
  30    ✓✗    1         set_padding_bits();
  31    ✓✓   49         for (uint8_t idx = 1; idx < 49; idx++)
  32                    {
  33    ✗✓   48             if (m_common_byte_register[idx] != 0) { built_in_test_fail++; }
  34                    }
  35                
  36                    // function bits test
  37                    // bits      [===]
  38                    //           =][==
  39                    // Bytes   #49  #50
  40                
  41    ✓✗    1         set_function_data(true, false, false, false, false);
  42    ✗✓    1         if (m_common_byte_register[49] != 0b00000010) built_in_test_fail++;   // 2
  43    ✓✗    1         set_function_data(true, true, false, false, false);
  44    ✗✓    1         if (m_common_byte_register[49] != 0b00000011) built_in_test_fail++;   // 3
  45    ✓✗    1         set_function_data(true, true, true, false, false);
  46    ✗✓    1         if (m_common_byte_register[50] != 0b10000000) built_in_test_fail++;   // 128
  47    ✓✗    1         set_function_data(true, true, true, true, false);
  48    ✗✓    1         if (m_common_byte_register[50] != 0b11000000) built_in_test_fail++;   // 192
  49    ✓✗    1         set_function_data(true, true, true, true, true);
  50    ✗✓    1         if (m_common_byte_register[50] != 0b11100000) built_in_test_fail++;   // 224
  51                
  52                    // BC bits test
  53                    // BC         blue   green   red
  54                    // bits      [=====][=====][=====]
  55                    // bits      ====][======][======]
  56                    // Bytes     #50    #51      #52
  57                
  58          1         std::bitset<m_bc_data_resolution> bc_test_on {127};
  59          1         std::bitset<m_bc_data_resolution> bc_test_off {0};
  60                
  61    ✗✓    1         if (m_common_byte_register[50] != 0b11100000) built_in_test_fail++;     // 224
  62    ✗✓    1         if (m_common_byte_register[51] != 0b00000000) built_in_test_fail++;
  63    ✗✓    1         if (m_common_byte_register[52] != 0b00000000) built_in_test_fail++;
  64                
  65    ✓✗    1         set_bc_data(bc_test_on, bc_test_off, bc_test_off);
  66    ✗✓    1         if (m_common_byte_register[50] != 0b11111111) built_in_test_fail++;     // 255
  67    ✗✓    1         if (m_common_byte_register[51] != 0b11000000) built_in_test_fail++;     // 192
  68    ✗✓    1         if (m_common_byte_register[52] != 0x00000000) built_in_test_fail++;
  69                
  70    ✓✗    1         set_bc_data(bc_test_off, bc_test_on, bc_test_off);
  71    ✗✓    1         if (m_common_byte_register[50] != 0b11100000) built_in_test_fail++;     // 224
  72    ✗✓    1         if (m_common_byte_register[51] != 0b00111111) built_in_test_fail++;     // 63
  73    ✗✓    1         if (m_common_byte_register[52] != 0b10000000) built_in_test_fail++;     // 128
  74                
  75    ✓✗    1         set_bc_data(bc_test_off, bc_test_off, bc_test_on);
  76    ✗✓    1         if (m_common_byte_register[50] != 0b11100000) built_in_test_fail++;     // 224
  77    ✗✓    1         if (m_common_byte_register[51] != 0x00000000) built_in_test_fail++;
  78    ✗✓    1         if (m_common_byte_register[52] != 0b01111111) built_in_test_fail++;     // 127
  79                
  80    ✓✗    1         set_bc_data(bc_test_off, bc_test_off, bc_test_off);
  81    ✗✓    1         if (m_common_byte_register[50] != 0b11100000) built_in_test_fail++;     // 224
  82    ✗✓    1         if (m_common_byte_register[51] != 0b00000000) built_in_test_fail++;
  83    ✗✓    1         if (m_common_byte_register[52] != 0b00000000) built_in_test_fail++;
  84                
  85    ✓✗    1         set_bc_data(bc_test_on, bc_test_on, bc_test_on);
  86    ✗✓    1         if (m_common_byte_register[50] != 0b11111111) built_in_test_fail++;     // 255
  87    ✗✓    1         if (m_common_byte_register[51] != 0b11111111) built_in_test_fail++;     // 255
  88    ✗✓    1         if (m_common_byte_register[52] != 0b11111111) built_in_test_fail++;     // 255
  89                
  90                    // MC         B  G  R
  91                    // bits      [=][=][=]
  92                    // bits      [======][
  93                    // Bytes       #53    #54
  94                
  95          1         std::bitset<m_mc_data_resolution> mc_test_on {7};
  96          1         std::bitset<m_mc_data_resolution> mc_test_off {0};
```

```
97  ✓✗  1      set_mc_data(mc_test_on, mc_test_off, mc_test_off);
98  ✗✓  1      if (m_common_byte_register[53] != 0b11100000) built_in_test_fail++;     // 224
99  ✗✓  1      if (m_common_byte_register[54] != 0b00000000) built_in_test_fail++;     // 0
100 ✓✗  1      set_mc_data(mc_test_off, mc_test_on, mc_test_off);
101 ✗✓  1      if (m_common_byte_register[53] != 0b00011100) built_in_test_fail++;     // 28
102 ✗✓  1      if (m_common_byte_register[54] != 0b00000000) built_in_test_fail++;     // 0
103 ✓✗  1      set_mc_data(mc_test_off, mc_test_off, mc_test_on);
104 ✗✓  1      if (m_common_byte_register[53] != 0b00000011) built_in_test_fail++;     // 3
105 ✗✓  1      if (m_common_byte_register[54] != 0b10000000) built_in_test_fail++;     // 128
106
107 ✓✗  1      set_mc_data(mc_test_off, mc_test_off, mc_test_off);
108 ✗✓  1      if (m_common_byte_register[53] != 0b00000000) built_in_test_fail++;
109 ✗✓  1      if (m_common_byte_register[54] != 0b00000000) built_in_test_fail++;
110
111     1      std::bitset<m_mc_data_resolution> mc_test_blue {1};
112     1      std::bitset<m_mc_data_resolution> mc_test_green {1};
113     1      std::bitset<m_mc_data_resolution> mc_test_red {1};
114 ✓✗  1      set_mc_data(mc_test_off, mc_test_off, mc_test_red);
115 ✗✓  1      if (m_common_byte_register[53] != 0b00000000) built_in_test_fail++;     // 0
116 ✗✓  1      if (m_common_byte_register[54] != 0b10000000) built_in_test_fail++;     // 128
117 ✓✗  1      set_mc_data(mc_test_off, mc_test_off, mc_test_red <<= 1);
118 ✗✓  1      if (m_common_byte_register[53] != 0b00000001) built_in_test_fail++;     // 1
119 ✗✓  1      if (m_common_byte_register[54] != 0b00000000) built_in_test_fail++;     // 0
120 ✓✗  1      set_mc_data(mc_test_off, mc_test_off, mc_test_red <<= 1);
121 ✗✓  1      if (m_common_byte_register[53] != 0b00000010) built_in_test_fail++;     // 2
122 ✗✓  1      if (m_common_byte_register[54] != 0b00000000) built_in_test_fail++;     // 0
123 ✓✗  1      set_mc_data(mc_test_off, mc_test_green, mc_test_red <<= 1);
124 ✗✓  1      if (m_common_byte_register[53] != 0b00000100) built_in_test_fail++;     // 4
125 ✗✓  1      if (m_common_byte_register[54] != 0b00000000) built_in_test_fail++;     // 0
126 ✓✗  1      set_mc_data(mc_test_off, mc_test_green <<= 1, mc_test_red);
127 ✗✓  1      if (m_common_byte_register[53] != 0b00001000) built_in_test_fail++;     // 8
128 ✗✓  1      if (m_common_byte_register[54] != 0b00000000) built_in_test_fail++;     // 0
129 ✓✗  1      set_mc_data(mc_test_off, mc_test_green <<= 1, mc_test_red);
130 ✗✓  1      if (m_common_byte_register[53] != 0b00010000) built_in_test_fail++;     // 16
131 ✗✓  1      if (m_common_byte_register[54] != 0b00000000) built_in_test_fail++;     // 0
132 ✓✗  1      set_mc_data(mc_test_blue, mc_test_green <<= 1, mc_test_red);
133 ✗✓  1      if (m_common_byte_register[53] != 0b00100000) built_in_test_fail++;     // 32
134 ✗✓  1      if (m_common_byte_register[54] != 0b00000000) built_in_test_fail++;     // 0
135 ✓✗  1      set_mc_data(mc_test_blue <<= 1, mc_test_green, mc_test_red);
136 ✗✓  1      if (m_common_byte_register[53] != 0b01000000) built_in_test_fail++;     // 64
137 ✗✓  1      if (m_common_byte_register[54] != 0b00000000) built_in_test_fail++;     // 0
138 ✓✗  1      set_mc_data(mc_test_blue <<= 1, mc_test_green, mc_test_red);
139 ✗✓  1      if (m_common_byte_register[53] != 0b10000000) built_in_test_fail++;     // 128
140 ✗✓  1      if (m_common_byte_register[54] != 0b00000000) built_in_test_fail++;     // 0
141
142
143
144     1      return built_in_test_fail;
145         }
146
147   629  void Driver::set_value_nth_bit(uint8_t &target, bool value, uint16_t shift_idx)
148         {
149 ✓✓  629      if (value) { target |= (1U << shift_idx); }
150   549      else { target &= ~(1U << shift_idx); }
151   629      print_common_bits();
152   629  }
153
154
155     1  void Driver::set_control_bit(bool ctrl_latch)
156         {
157             // Latch
158             // bits      =
159             // Bytes     [
160             //          #0
161
162             //m_common_bit_register.set(m_latch_offset, ctrl_latch);
163     1       set_value_nth_bit(m_common_byte_register[0], ctrl_latch, 7);
164     1   }
165
166     1  void Driver::set_ctrl_cmd_bits()
167         {
168
169             // Ctrl      10010110
170             // bits      [======]
171             // Bytes     ======][
172             //             #0    #1
173
174             // 7 MSB bits of ctrl byte into 7 LSB of byte #0
175 ✓✓  8       for (int8_t idx = m_ctrl_cmd_size_bits - 1; idx > 0; idx--)
176             {
177     7           set_value_nth_bit(m_common_byte_register[0], m_ctrl_cmd.test(idx), idx -1 );
178
179             }
180
181             // the last m_ctrl_cmd bit in to MSB of byte #1
182     1       set_value_nth_bit(m_common_byte_register[1], m_ctrl_cmd.test(0), 7);
183
184     1   }
185
186     1  void Driver::set_padding_bits()
187         {
188
189             // Padding  0  ================================================================================ 79
190             // Bytes       ======][======][======][======][======][======][======][======][======][======][
191             //               #1      #2       #3     #4       #5       #6       #7       #8       #9      #10
192
193             // Padding 80  ================================================================================ 159
194             // Bytes       ======][======][======][======][======][======][======][======][======][======][
195             //              #11     #12      #13     #14      #15      #16      #17      #18      #19      #20
196
197             // Padding 160 ================================================================================ 239
198             // Bytes       ======][======][======][======][======][======][======][======][======][======][
199             //              #21     #22      #23     #24      #25      #26      #27      #28      #29      #30
200
201             // Padding 240 ================================================================================ 319
```

```
202                 // Bytes       ======][======][======][======][======][======][======][======][======][
203                 //              #31     #32     #33     #34     #35     #36     #37     #38     #39     #40
204
205                 // Padding 320 ================================================================== 389
206                 // Bytes       ======][======][======][======][======][======][======][======][=====
207                 //              #41     #42     #43     #44     #45     #46     #47     #48     #49
208
209                 // first, we write 7 LSB bits of m_common_byte_register[1] = 0
210    ✓✓     8     for (int8_t idx = 6; idx > -1; idx--)
211                 {
212            7        set_value_nth_bit(m_common_byte_register[1], false, idx);
213                 }
214
215                 // The next 47 bytes are don't care padding = 0
216            1    const uint16_t padding_bytes_remaining = 47U;
217    ✓✓     46    for (uint16_t byte_idx  = 2; byte_idx < padding_bytes_remaining; byte_idx++)
218                 {
219    ✓✓     405       for (int8_t bit_idx = 7; bit_idx > -1; bit_idx--)
220                      {
221            360           set_value_nth_bit(m_common_byte_register[byte_idx], false, bit_idx);
222                      }
223                 }
224
225                 // lastly, we write 6 MSB bits of m_common_byte_register[49] = 0
226    ✓✓     7     for (int8_t idx = 7; idx > 1; idx--)
227                 {
228            6        set_value_nth_bit(m_common_byte_register[49], false, idx);
229                 }
230
231            1 }
232
233            5 void Driver::set_function_data(bool DSPRPT, bool TMGRST, bool RFRESH, bool ESPWM, bool LSDVLT)
234              {
235
236                 // Function
237                 // bits      [===]
238                 //           =][==
239                 // Bytes   #49  #50
240
241                 // if all are set to true, byte #49 = 3, byte #50 = 224
242            5    set_value_nth_bit(m_common_byte_register[49], DSPRPT, 1);
243            5    set_value_nth_bit(m_common_byte_register[49], TMGRST, 0);
244            5    set_value_nth_bit(m_common_byte_register[50], RFRESH, 7);
245            5    set_value_nth_bit(m_common_byte_register[50], ESPWM, 6);
246            5    set_value_nth_bit(m_common_byte_register[50], LSDVLT, 5);
247            5 }
248
249            5 void Driver::set_bc_data(std::bitset<m_bc_data_resolution> &blue_value,
250                 std::bitset<m_bc_data_resolution> &green_value,
251                 std::bitset<m_bc_data_resolution> &red_value)
252              {
253                 // BC         blue   green   red
254                 // bits      [=====][=====][=====]
255                 // bits      ====][======][======]
256                 // Bytes     #50    #51      #52
257
258                 // set 5 LSB of byte #50 to bits 6-2 of BC blue_value
259    ✓✓     30    for (int8_t bit_idx = m_bc_data_resolution - 1; bit_idx > 1; bit_idx--)
260                 {
261                    // offset the bit position in byte #50 by 2 places.
262            25      set_value_nth_bit(m_common_byte_register[50], blue_value.test(bit_idx), bit_idx - 2);
263                 }
264
265                 // set the first 2 MSB bits of byte #51 to the last 2 LSB of blue_value
266            5    set_value_nth_bit(m_common_byte_register[51], blue_value.test(1), 7);
267            5    set_value_nth_bit(m_common_byte_register[51], blue_value.test(0), 6);
268
269                 // set 5 LSB of byte #51 to bits 6-1 of BC green_value
270    ✓✓     35    for (int8_t bit_idx = m_bc_data_resolution - 1; bit_idx > 0; bit_idx--)
271                 {
272                    // offset the bit position in byte #51 by 1 places.
273            30      set_value_nth_bit(m_common_byte_register[51], green_value.test(bit_idx), bit_idx - 1);
274                 }
275
276                 // set MSB of byte#52 to LSB of green_value
277            5    set_value_nth_bit(m_common_byte_register[52], green_value.test(0), 7);
278
279                 // set 7 LSB of byte #50 to bits all 7 bits of BC red_value
280    ✓✓     40    for (int8_t bit_idx = m_bc_data_resolution - 1; bit_idx > -1; bit_idx--)
281                 {
282                    // No offset for bit position in byte #52.
283            35      set_value_nth_bit(m_common_byte_register[52], red_value.test(bit_idx), bit_idx);
284                 }
285
286            5 }
287
288           13 void Driver::set_mc_data(std::bitset<m_mc_data_resolution> &blue_value,
289                 std::bitset<m_mc_data_resolution> green_value,
290                 std::bitset<m_mc_data_resolution> &red_value)
291              {
292                 // MC          B  G  R
293                 // bits       [=][=][=]
294                 // bits       [======][
295                 // Bytes        #53    #54
296
297                 // 3 bits of blue in 3 MSB of byte #51 == 128
298           13    set_value_nth_bit(m_common_byte_register[53], blue_value.test(m_mc_data_resolution - 1), 7);
299           13    set_value_nth_bit(m_common_byte_register[53], blue_value.test(m_mc_data_resolution - 2), 6);
300           13    set_value_nth_bit(m_common_byte_register[53], blue_value.test(m_mc_data_resolution - 3), 5);
301
302                 // 3 bits of green in next 3 bits of byte #51 == 144
303           13    set_value_nth_bit(m_common_byte_register[53], green_value.test(m_mc_data_resolution - 1), 4);
304           13    set_value_nth_bit(m_common_byte_register[53], green_value.test(m_mc_data_resolution - 2), 3);
305           13    set_value_nth_bit(m_common_byte_register[53], green_value.test(m_mc_data_resolution - 3), 2);
306
307                 // 3 bits of red in 2 LSB of byte #51 (== 146) and MSB of byte #52 (== 0)
```

```
    set_value_nth_bit(m_common_byte_register[53], red_value.test(m_mc_data_resolution - 1), 1);
    set_value_nth_bit(m_common_byte_register[53], red_value.test(m_mc_data_resolution - 2), 0);
    set_value_nth_bit(m_common_byte_register[54], red_value.test(m_mc_data_resolution - 3), 7);

}

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

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

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

    // 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



    switch(led_idx)
    {
        case 0:

            set_value_nth_bit(m_common_byte_register[93], blue_value.test(6), 3);
            set_value_nth_bit(m_common_byte_register[93], blue_value.test(5), 2);
            set_value_nth_bit(m_common_byte_register[93], blue_value.test(4), 1);
            set_value_nth_bit(m_common_byte_register[93], blue_value.test(3), 0);
            set_value_nth_bit(m_common_byte_register[94], blue_value.test(2), 7);
            set_value_nth_bit(m_common_byte_register[94], blue_value.test(1), 6);
            set_value_nth_bit(m_common_byte_register[94], blue_value.test(0), 5);

            set_value_nth_bit(m_common_byte_register[94], green_value.test(6), 4);
            set_value_nth_bit(m_common_byte_register[94], green_value.test(5), 3);
            set_value_nth_bit(m_common_byte_register[94], green_value.test(4), 2);
            set_value_nth_bit(m_common_byte_register[94], green_value.test(3), 1);
            set_value_nth_bit(m_common_byte_register[94], green_value.test(2), 0);
            set_value_nth_bit(m_common_byte_register[95], green_value.test(1), 7);
            set_value_nth_bit(m_common_byte_register[95], green_value.test(0), 6);

            set_value_nth_bit(m_common_byte_register[95], red_value.test(6), 5);
            set_value_nth_bit(m_common_byte_register[95], red_value.test(5), 4);
            set_value_nth_bit(m_common_byte_register[95], red_value.test(4), 3);
            set_value_nth_bit(m_common_byte_register[95], red_value.test(3), 2);
            set_value_nth_bit(m_common_byte_register[95], red_value.test(2), 1);
            set_value_nth_bit(m_common_byte_register[95], red_value.test(1), 0);
            set_value_nth_bit(m_common_byte_register[96], red_value.test(0), 7);

            break;



        case 1:
            set_value_nth_bit(m_common_byte_register[90], blue_value.test(6), 0);
            set_value_nth_bit(m_common_byte_register[91], blue_value.test(5), 7);
            set_value_nth_bit(m_common_byte_register[91], blue_value.test(4), 6);
            set_value_nth_bit(m_common_byte_register[91], blue_value.test(3), 5);
            set_value_nth_bit(m_common_byte_register[91], blue_value.test(2), 4);
            set_value_nth_bit(m_common_byte_register[91], blue_value.test(1), 3);
            set_value_nth_bit(m_common_byte_register[91], blue_value.test(0), 2);

            set_value_nth_bit(m_common_byte_register[91], green_value.test(6), 1);
            set_value_nth_bit(m_common_byte_register[91], green_value.test(5), 0);
            set_value_nth_bit(m_common_byte_register[92], green_value.test(4), 7);
            set_value_nth_bit(m_common_byte_register[92], green_value.test(3), 6);
            set_value_nth_bit(m_common_byte_register[92], green_value.test(2), 5);
            set_value_nth_bit(m_common_byte_register[92], green_value.test(1), 4);
            set_value_nth_bit(m_common_byte_register[92], green_value.test(0), 3);

            set_value_nth_bit(m_common_byte_register[92], red_value.test(6), 2);
            set_value_nth_bit(m_common_byte_register[92], red_value.test(5), 1);
            set_value_nth_bit(m_common_byte_register[92], red_value.test(4), 0);
            set_value_nth_bit(m_common_byte_register[93], red_value.test(3), 7);
            set_value_nth_bit(m_common_byte_register[93], red_value.test(2), 6);
            set_value_nth_bit(m_common_byte_register[93], red_value.test(1), 5);
            set_value_nth_bit(m_common_byte_register[93], red_value.test(0), 4);

            break;
        case 2:

            set_value_nth_bit(m_common_byte_register[88], blue_value.test(6), 5);
            set_value_nth_bit(m_common_byte_register[88], blue_value.test(5), 4);
            set_value_nth_bit(m_common_byte_register[88], blue_value.test(4), 3);
            set_value_nth_bit(m_common_byte_register[88], blue_value.test(3), 2);
            set_value_nth_bit(m_common_byte_register[88], blue_value.test(2), 1);
            set_value_nth_bit(m_common_byte_register[88], blue_value.test(1), 0);
            set_value_nth_bit(m_common_byte_register[89], blue_value.test(0), 7);

            set_value_nth_bit(m_common_byte_register[89], green_value.test(6), 6);
            set_value_nth_bit(m_common_byte_register[89], green_value.test(5), 5);
            set_value_nth_bit(m_common_byte_register[89], green_value.test(4), 4);
            set_value_nth_bit(m_common_byte_register[89], green_value.test(3), 3);
            set_value_nth_bit(m_common_byte_register[89], green_value.test(2), 2);
            set_value_nth_bit(m_common_byte_register[89], green_value.test(1), 1);
            set_value_nth_bit(m_common_byte_register[89], green_value.test(0), 0);

            set_value_nth_bit(m_common_byte_register[90], red_value.test(6), 7);
            set_value_nth_bit(m_common_byte_register[90], red_value.test(5), 6);
            set_value_nth_bit(m_common_byte_register[90], red_value.test(4), 5);
```

```
            set_value_nth_bit(m_common_byte_register[90], red_value.test(3), 4);
            set_value_nth_bit(m_common_byte_register[90], red_value.test(2), 3);
            set_value_nth_bit(m_common_byte_register[90], red_value.test(1), 2);
            set_value_nth_bit(m_common_byte_register[90], red_value.test(0), 1);

            break;
        case 3:

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

            set_value_nth_bit(m_common_byte_register[85], blue_value.test(6), 2);
            set_value_nth_bit(m_common_byte_register[85], blue_value.test(5), 1);
            set_value_nth_bit(m_common_byte_register[85], blue_value.test(4), 0);
            set_value_nth_bit(m_common_byte_register[86], blue_value.test(3), 7);
            set_value_nth_bit(m_common_byte_register[85], blue_value.test(2), 6);
            set_value_nth_bit(m_common_byte_register[85], blue_value.test(1), 5);
            set_value_nth_bit(m_common_byte_register[85], blue_value.test(0), 4);









            break;
        case 4:
            break;
        case 5:
            break;
        case 6:
            break;
        case 7:
            break;
        case 8:
            break;
        case 9:
            break;
        case 10:
            break;
        case 11:
            break;
        case 12:
            break;
        case 13:
            break;
        case 14:
            break;
        case 15:
            break;

    }

}

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

void Driver::set_gs_data(uint8_t led_pos, std::bitset<16> &blue_value, std::bitset<16> &green_value, std::bitset<16> &red_value)
{
    // offset for the current LED position
    const uint16_t led_offset = m_gs_data_one_led_size_bits * led_pos;

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

    // add each blue_value bit into the BC section of the common register
    for (uint8_t idx = 0; idx < blue_value.size(); idx++)
    {
        // make sure we stay within bounds of the common register
        if (gs_common_pos < m_common_reg_size_bits)
        {
           m_common_bit_register.set(gs_common_pos, blue_value[idx]);
           gs_common_pos++;
        }
    }

    // add each green_value bit into the GS section of the common register
    for (uint8_t idx = 0; idx < green_value.size(); idx++)
    {
        // make sure we stay within bounds of the common register
        if (gs_common_pos < m_common_reg_size_bits)
        {
           m_common_bit_register.set(gs_common_pos, green_value[idx]);
           gs_common_pos++;
        }
    }

    // add each red_value bit into the GS section of the common register
    for (uint8_t idx = 0; idx < red_value.size(); idx++)
    {
        // make sure we stay within bounds of the common register
        if (gs_common_pos < m_common_reg_size_bits)
        {
           m_common_bit_register.set(gs_common_pos, red_value[idx]);
           gs_common_pos++;
        }
    }
}

void Driver::set_all_gs_data(std::bitset<m_gs_data_resolution> &blue_value,
```

```
524         std::bitset<m_gs_data_resolution> &green_value,
525         std::bitset<m_gs_data_resolution> &red_value)
526     {
527         for (uint8_t led_idx = 0; led_idx < m_num_leds_per_chip; led_idx++)
528         {
529             set_gs_data(led_idx, blue_value, green_value, red_value);
530         }
531     }
532
533
534
535     void Driver::send_data()
536     {
537         // clock the data through and latch
538     #ifdef USE_HAL_DRIVER
539         HAL_StatusTypeDef res = HAL_SPI_Transmit(&m_spi_interface, (uint8_t*)m_common_byte_register.data(), m_common_reg_size_bytes, HAL_MAX_DELAY);
540         UNUSED(res);
541     #endif
542         toggle_latch();
543     }
544
545     void Driver::toggle_latch()
546     {
547     #ifdef USE_HAL_DRIVER
548         HAL_Delay(m_latch_delay_ms);
549         HAL_GPIO_WritePin(m_lat_port, m_lat_pin, GPIO_PIN_SET);
550         HAL_Delay(m_latch_delay_ms);
551         HAL_GPIO_WritePin(m_lat_port, m_lat_pin, GPIO_PIN_RESET);
552         HAL_Delay(m_latch_delay_ms);
553     #endif
554     }
555
556     void Driver::flush_common_register()
557     {
558         m_common_bit_register.reset();
559         send_data();
560     }
561
562 629 void Driver::print_common_bits()
563     {
564     #ifdef USE_RTT
565         SEGGER_RTT_printf(0, "\r\n");
566         for (uint16_t idx = 45; idx < 53; idx++)
567         {
568             SEGGER_RTT_printf(0, "%u ", +m_common_byte_register[idx]);
569         }
570     #endif
571 629 }
572
573     // void Driver::flush_common_register()
574     // {
575     //     // reset the latch
576     //     HAL_GPIO_WritePin(m_lat_port, m_lat_pin, GPIO_PIN_RESET);
577
578     //     // clock-in the entire common shift register per daisy-chained chip before pulsing the latch
579     //     for (uint8_t shift_entire_reg = 0; shift_entire_reg < m_num_driver_ics; shift_entire_reg++)
580     //     {
581     //         // write the MSB bit low to signal greyscale data
582     //         HAL_GPIO_WritePin(m_sck_port, m_sck_pin, GPIO_PIN_RESET);
583     //         HAL_GPIO_WritePin(m_mosi_port, m_mosi_pin, GPIO_PIN_RESET);
584     //         HAL_GPIO_WritePin(m_sck_port, m_sck_pin, GPIO_PIN_SET);
585     //         HAL_GPIO_WritePin(m_sck_port, m_sck_pin, GPIO_PIN_RESET);
586
587     //         // Set all 16-bit colours to 0 greyscale
588     //         uint8_t grayscale_data[2] = {0x00, 0x00};
589     //         for (uint8_t idx = 0; idx < 16; idx++)
590     //         {
591     //             HAL_SPI_Transmit(&m_spi_interface, grayscale_data, 2, HAL_MAX_DELAY);
592     //             HAL_SPI_Transmit(&m_spi_interface, grayscale_data, 2, HAL_MAX_DELAY);
593     //             HAL_SPI_Transmit(&m_spi_interface, grayscale_data, 2, HAL_MAX_DELAY);
594     //         }
595     //     }
596
597     //     toggle_latch();
598     // }
599
600     // void Driver::enable_spi()
601     // {
602     //     HAL_GPIO_DeInit(GPIOB, TLC5955_SPI2_MOSI_Pin|TLC5955_SPI2_SCK_Pin);
603
604     //     m_spi_interface.Instance = SPI2;
605     //     m_spi_interface.Init.Mode = SPI_MODE_MASTER;
606     //     m_spi_interface.Init.Direction = SPI_DIRECTION_1LINE;
607     //     m_spi_interface.Init.DataSize = SPI_DATASIZE_8BIT;
608     //     m_spi_interface.Init.CLKPolarity = SPI_POLARITY_LOW;
609     //     m_spi_interface.Init.CLKPhase = SPI_PHASE_1EDGE;
610     //     m_spi_interface.Init.NSS = SPI_NSS_SOFT;
611     //     m_spi_interface.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_8;
612     //     m_spi_interface.Init.FirstBit = SPI_FIRSTBIT_MSB;
613     //     m_spi_interface.Init.TIMode = SPI_TIMODE_DISABLE;
614     //     m_spi_interface.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
615     //     m_spi_interface.Init.CRCPolynomial = 7;
616     //     m_spi_interface.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
617     //     m_spi_interface.Init.NSSPMode = SPI_NSS_PULSE_DISABLE;
618
619     //     if (HAL_SPI_Init(&m_spi_interface) != HAL_OK) { Error_Handler(); }
620
621     //     __HAL_RCC_SPI2_CLK_ENABLE();
622     //     __HAL_RCC_GPIOB_CLK_ENABLE();
623
624     //     GPIO_InitTypeDef GPIO_InitStruct = {
625     //         TLC5955_SPI2_MOSI_Pin|TLC5955_SPI2_SCK_Pin,
626     //         GPIO_MODE_AF_PP,
627     //         GPIO_PULLDOWN,
628     //         GPIO_SPEED_FREQ_VERY_HIGH,
629     //         GPIO_AF1_SPI2,
630     //     };
631
```

```
632         //     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
633         
634         //     __HAL_SYSCFG_FASTMODEPLUS_ENABLE(SYSCFG_FASTMODEPLUS_PB8);
635         
636         // }
637         
638         // void Driver::disable_spi()
639         // {
640         
641         // }
642         
643         // void Driver::enable_gpio_output_only()
644         // {
645         //     // disable SPI config
646         //     __HAL_RCC_SPI2_CLK_DISABLE();
647         //     HAL_GPIO_DeInit(GPIOB, TLC5955_SPI2_MOSI_Pin|TLC5955_SPI2_SCK_Pin);
648         
649         //     // GPIO Ports Clock Enable
650         //     __HAL_RCC_GPIOB_CLK_ENABLE();
651         
652         //     // Configure GPIO pin Output Level
653         //     HAL_GPIO_WritePin(GPIOB, TLC5955_SPI2_LAT_Pin|TLC5955_SPI2_GSCLK_Pin|TLC5955_SPI2_MOSI_Pin|TLC5955_SPI2_SCK_Pin, GPIO_PIN_RESET);
654         
655         //     // Configure GPIO pins
656         //     GPIO_InitTypeDef GPIO_InitStruct = {
657         //         TLC5955_SPI2_LAT_Pin|TLC5955_SPI2_GSCLK_Pin|TLC5955_SPI2_MOSI_Pin|TLC5955_SPI2_SCK_Pin,
658         //         GPIO_MODE_OUTPUT_PP,
659         //         GPIO_PULLDOWN,
660         //         GPIO_SPEED_FREQ_VERY_HIGH,
661         //         0
662         //     };
663         
664         //     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
665         
666         //     __HAL_SYSCFG_FASTMODEPLUS_ENABLE(SYSCFG_FASTMODEPLUS_PB9);
667         //     __HAL_SYSCFG_FASTMODEPLUS_ENABLE(SYSCFG_FASTMODEPLUS_PB6);
668         //     __HAL_SYSCFG_FASTMODEPLUS_ENABLE(SYSCFG_FASTMODEPLUS_PB7);
669         //     __HAL_SYSCFG_FASTMODEPLUS_ENABLE(SYSCFG_FASTMODEPLUS_PB8);
670         
671         // }
672         
673         } // namespace tlc5955
```

Generated by: GCOVR (Version 4.2)

GCC Code Coverage Report

		Exec	Total	Coverage
Directory:	./			
Date:	2021-11-28 15:51:03	**Lines:**	297	445	66.7 %
Legend:	low: < 75.0 % medium: >= 75.0 % high: >= 90.0 %	**Branches:**	179	415	43.1 %

File	Lines			Branches	
cpp_ssd1306/inc/font.hpp		100.0 %	8 / 8	100.0 %	2 / 2
cpp_ssd1306/inc/ssd1306.hpp		86.3 %	44 / 51	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 %	33 / 33	56.8 %	25 / 44
cpp_ssd1306/tests/ssd1306_tester.hpp		100.0 %	4 / 4	- %	0 / 0
cpp_tlc5955/src/tlc5955.cpp		52.2 %	143 / 274	45.6 %	99 / 217
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:	2021-11-28 15:51:03	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            0    void mainapp()
  24                 {
  25
  26                  static ssd1306::Font5x7 font;
  27            0     static ssd1306::Display oled;
  28            0     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            0     uint8_t count = 0;
  57
  58                  while(true)
  59                  {
  60
  61            0      std::stringstream msg;
  62            0      msg << font.character_map[count];
  63            0      oled.write(msg, font, 2, 2, ssd1306::Colour::Black, ssd1306::Colour::White, 3, true);
  64            0      if (count < font.character_map.size() - 1) { count++; }
  65            0      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
```