GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	src/	**Lines:**	0	45	0.0 %
Date:	2022-03-19 23:25:32	**Branches:**	0	26	0.0 %
Legend:	low: < 75.0 % medium: >= 75.0 % high: >= 90.0 %				

File	Lines			Branches	
isr_manager_stm32g0.cpp		0.0 %	0 / 45	0.0 %	0 / 26

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	src/	**Lines:**	0	45	0.0 %
Date:	2022-03-19 23:25:32	**Branches:**	0	26	0.0 %
Legend:	low: < 75.0 % medium: >= 75.0 % high: >= 90.0 %				

File	Lines			Branches	
isr_manager_stm32g0.cpp		0.0 %	0 / 45	0.0 %	0 / 26

GCC Code Coverage Report

			Exec	Total	Coverage
Directory:	src/				
File:	src/isr_manager_stm32g0.cpp	Lines:	0	45	0.0 %
Date:	2022-03-19 23:25:32	Branches:	0	26	0.0 %

```
Line Branch Exec  Source
   1                // MIT License
   2
   3                // Copyright (c) 2022 Chris Sutton
   4
   5                // Permission is hereby granted, free of charge, to any person obtaining a copy
   6                // of this software and associated documentation files (the "Software"), to deal
   7                // in the Software without restriction, including without limitation the rights
   8                // to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
   9                // copies of the Software, and to permit persons to whom the Software is
  10                // furnished to do so, subject to the following conditions:
  11
  12                // The above copyright notice and this permission notice shall be included in all
  13                // copies or substantial portions of the Software.
  14
  15                // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
  16                // IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
  17                // FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
  18                // AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
  19                // LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
  20                // OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
  21                // SOFTWARE.
  22
  23
  24                #include <isr_manager_stm32g0.hpp>
  25
  26
  27
  28                namespace stm32::isr
  29                {
  30
  31    #####       extern "C" void EXTI4_15_IRQHandler(void)
  32                {
  33                    // This calls all registered (non-null) ISR functions. Add more as needed. Must add the following:
  34                    // 1. check if EXTI flag is set before calling ISR function (using LL_EXTI_IsActiveFallingFlag_0_31)
  35                    // 2. check ISR slot has a callback function set (not nullptr)
  36                    // 3. reset the interrupt flag for their EXTI (using LL_EXTI_ClearFallingFlag_0_31)
  37    #####           if ( (EXTI->FPR1 & EXTI_IMR1_IM5) == EXTI_IMR1_IM5 )
  38                    {
  39    #####               if (InterruptManagerStm32g0::m_interrupt_handlers[ static_cast<int>( InterruptTypeStm32g0::exti5 ) ] != nullptr)
  40                        {
  41    #####                   InterruptManagerStm32g0::m_interrupt_handlers[ static_cast<int>( InterruptTypeStm32g0::exti5 ) ]->ISR();
  42                            // clear the falling flag for EXTI Line 5
  43    #####                   EXTI->FPR1 = EXTI->FPR1 | EXTI_IMR1_IM5;
  44                        }
  45                        else
  46                        {
  47    #####                   while(true) { /* No ISR registered in InterruptManagerStm32g0 */ }
  48                        }
  49                    }
  50    #####           else if ( (EXTI->FPR1 & EXTI_IMR1_IM10) == EXTI_IMR1_IM10 )
  51                    {
  52    #####               if (InterruptManagerStm32g0::m_interrupt_handlers[ static_cast<int>( InterruptTypeStm32g0::exti10 ) ] != nullptr)
  53                        {
  54    #####                   InterruptManagerStm32g0::m_interrupt_handlers[ static_cast<int>( InterruptTypeStm32g0::exti10 ) ]->ISR();
  55                            // clear the falling flag for EXTI Line 10
  56    #####                   EXTI->FPR1 = EXTI->FPR1 | EXTI_IMR1_IM10;
  57
  58                        }
  59                        else
  60                        {
  61    #####                   while(true) { /* No ISR registered in InterruptManagerStm32g0 */ }
  62                        }
  63                    }
  64
  65
  66                }
  67
  68    #####       extern "C" void DMA1_Channel1_IRQHandler(void)
  69                {
  70    #####           if (InterruptManagerStm32g0::m_interrupt_handlers[ static_cast<int>( InterruptTypeStm32g0::dma1_ch2 ) ] != nullptr)
  71                    {
  72    #####               InterruptManagerStm32g0::m_interrupt_handlers[ static_cast<int>( InterruptTypeStm32g0::dma1_ch2 ) ]->ISR();
  73                    }
  74                    else
  75                    {
  76    #####               while(true) { /* No ISR registered in InterruptManagerStm32g0 */ }
  77                    }
  78                }
  79
  80
  81
  82    #####       extern "C" void TIM2_IRQHandler(void)
  83                {
```

```
        if (InterruptManagerStm32g0::m_interrupt_handlers[ static_cast<int>( InterruptTypeStm32g0::tim2 ) ] != nullptr)
        {
            InterruptManagerStm32g0::m_interrupt_handlers[ static_cast<int>( InterruptTypeStm32g0::tim2 ) ]->ISR();
        }
        else
        {
            while(true) { /* No ISR registered in InterruptManagerStm32g0 */ }
        }
    }

    extern "C" void TIM3_TIM4_IRQHandler(void)
    {
        if (InterruptManagerStm32g0::m_interrupt_handlers[ static_cast<int>( InterruptTypeStm32g0::tim3 ) ] != nullptr)
        {
            InterruptManagerStm32g0::m_interrupt_handlers[ static_cast<int>( InterruptTypeStm32g0::tim3 ) ]->ISR();
        }
        else if (InterruptManagerStm32g0::m_interrupt_handlers[ static_cast<int>( InterruptTypeStm32g0::tim4 ) ] != nullptr)
        {
            InterruptManagerStm32g0::m_interrupt_handlers[ static_cast<int>( InterruptTypeStm32g0::tim4 ) ]->ISR();
        }
        else
        {
            while(true) { /* No ISR registered in InterruptManagerStm32g0 */ }
        }
    }

    extern "C" void TIM14_IRQHandler(void)
    {
        if (InterruptManagerStm32g0::m_interrupt_handlers[ static_cast<int>( InterruptTypeStm32g0::tim14 ) ] != nullptr)
        {
            InterruptManagerStm32g0::m_interrupt_handlers[ static_cast<int>( InterruptTypeStm32g0::tim14 ) ]->ISR();
        }
        else
        {
            while(true) { /* No ISR registered in InterruptManagerStm32g0 */ }
        }

    }

    extern "C" void TIM15_IRQHandler(void)
    {
        if (InterruptManagerStm32g0::m_interrupt_handlers[ static_cast<int>( InterruptTypeStm32g0::tim15 ) ] != nullptr)
        {
            InterruptManagerStm32g0::m_interrupt_handlers[ static_cast<int>( InterruptTypeStm32g0::tim15 ) ]->ISR();
        }
        else
        {
            while(true) { /* No ISR registered in InterruptManagerStm32g0 */ }
        }

    }

    extern "C" void TIM16_FDCAN_IT0_IRQHandler(void)
    {
        if (InterruptManagerStm32g0::m_interrupt_handlers[ static_cast<int>( InterruptTypeStm32g0::tim16 ) ] != nullptr)
        {
            InterruptManagerStm32g0::m_interrupt_handlers[ static_cast<int>( InterruptTypeStm32g0::tim16 ) ]->ISR();
        }
        else
        {
            while(true) { /* No ISR registered in InterruptManagerStm32g0 */ }
        }

    }

    extern "C" void TIM7_LPTIM2_IRQHandler(void)
    {
        if (InterruptManagerStm32g0::m_interrupt_handlers[ static_cast<int>( InterruptTypeStm32g0::tim7 ) ] != nullptr)
        {
            InterruptManagerStm32g0::m_interrupt_handlers[ static_cast<int>( InterruptTypeStm32g0::tim7 ) ]->ISR();
        }
        else
        {
            while(true) { /* No ISR registered in InterruptManagerStm32g0 */ }
        }
    }

    extern "C" void USART3_4_5_6_LPUART1_IRQHandler(void)
    {
        if (InterruptManagerStm32g0::m_interrupt_handlers[ static_cast<int>( InterruptTypeStm32g0::usart5 ) ] != nullptr)
        {
            InterruptManagerStm32g0::m_interrupt_handlers[ static_cast<int>( InterruptTypeStm32g0::usart5 ) ]->ISR();
        }
        else
        {
            while(true) { /* No ISR registered in InterruptManagerStm32g0 */ }
        }
    }




} // namespace stm32::isr
```

176		