Murasaki Class Library 0.1.0

Generated by Doxygen 1.8.11

# **Contents**

1	Pref	ace	1
	1.1	Simplified IO	1
	1.2	Preemptive multi-task	2
	1.3	Blocking IO	2
	1.4	Thread safe IO	2
	1.5	Versatile printf() logger	2
	1.6	Guard by assertion	3
	1.7	System Logging	3
	1.8	Configurable	3
2	Targ	et and Environment	5
3	Usaç	ge Introduction	7
	3.1	Message output	7
	3.2	Serial communication	8
	3.3	Debugging with Murasaki	8
	3.4	Tasking	10
	3.5	Other peripheral	10
		3.5.1 I2C Master	11
		3.5.2 I2C Slave	11
		3.5.3 SPI Master	11
		3.5.4 SPI Slave	12

iv CONTENTS

4	Port	ing guid	ide	13
	4.1	Directo	tory Structure	13
		4.1.1	Src directory	14
		4.1.2	Inc directory	14
		4.1.3	Src-tp and Inc-tp directory	14
		4.1.4	murasaki.hpp	14
		4.1.5	template directory	14
			4.1.5.1 platform_config.hpp	14
			4.1.5.2 platform_defs.hpp	14
			4.1.5.3 murasaki_platform.hpp	15
			4.1.5.4 murasaki_platform.cpp	15
	4.2	CubeN	MX setting	15
		4.2.1	Heap Size	15
		4.2.2	Stack Size	16
		4.2.3	Task stack size of the default task	16
	4.3	Config	guration	16
	4.4	Task P	Priority and Stack Size	17
	4.5	Heap r	memory consideration	17
	4.6	Platfor	rm variable	18
	4.7	Routin	ng interrupts	19
	4.8	Error h	handling	20
	4.9	Summ	nary of the porting	21
5	Mod	ule Inde	lev.	23
•	5.1		les	
	5.1	Module	les	23
6	Nam	espace	e Index	25
	6.1	Names	espace List	25
7	Hiera	archica	al Index	27
	7.1		Hierarchy	27

CONTENTS

8	Clas	s Index															29
	8.1	Class I	₋ist											 	 	 	29
9	File I	ndex															31
	9.1	File Lis	t											 	 	 	31
10	Mod	ule Doc	umentatio	on													35
	10.1	Murasa	aki Class C	Collection	n									 	 	 	35
		10.1.1	Detailed	Descrip	tion									 	 	 	36
		10.1.2	Macro De	efinition	Docur	menta	ition							 	 	 	36
			10.1.2.1	MURA	\SAKI_	_ASSI	ERT .							 	 	 	36
			10.1.2.2	MURA	\SAKI_	_PRIN	NT_ER	RROR						 	 	 	37
			10.1.2.3	MURA	\SAKI_	_SYSI	LOG							 	 	 	37
	10.2	Synchr	onization a	and Exc	lusive	acces	SS .							 	 	 	39
		10.2.1	Detailed	Descrip	tion									 	 	 	39
	10.3	Third p	arty classe	es										 	 	 	40
		10.3.1	Detailed	Descrip	tion									 	 	 	40
	10.4	Definiti	ons and C	onfigur	ation									 	 	 	41
		10.4.1	Detailed	Descrip	tion									 	 	 	41
		10.4.2	Macro De	efinition	Docur	menta	ition							 	 	 	41
			10.4.2.1	MURA	\SAKI_	_CON	IFIG_N	NODE	BUG					 	 	 	41
			10.4.2.2	PLATI	FORM	_CON	NFIG_	DEBU	G_BI	UFFE	ER_S	SIZE		 	 	 	41
			10.4.2.3	PLATI	FORM	_CON	NFIG_	DEBU	G_LI	NE_S	SIZE			 	 	 	41
			10.4.2.4	PLATI	FORM	_CON	NFIG_	DEBU	G_SI	ERIA	L_TI	MEC	UT	 	 	 	42
			10.4.2.5	PLATI	FORM	_CON	NFIG_	DEBU	G_T/	ASK_	PRI	ORIT	Υ.	 	 	 	42
			10.4.2.6	PLATI	FORM	_CON	NFIG_	DEBU	G_T/	ASK_	_STA	CK_	SIZE	 	 	 	42
		10.4.3	Enumera	tion Typ	e Doc	umen	ıtation							 	 	 	42
			10.4.3.1	I2cSta	ıtus .									 	 	 	42
			10.4.3.2	SpiClo	ockPha	ase.								 	 	 	43
			10.4.3.3	SpiClo	ockPola	arity								 	 	 	43
			10.4.3.4	SpiSta	atus .									 	 	 	43

vi

10.4.3.5 SyslogFacility	44
10.4.3.6 SyslogSeverity	44
10.4.3.7 UartHardwareFlowControl	45
10.4.3.8 UartStatus	45
10.4.3.9 UartTimeout	45
10.4.3.10 WaitMilliSeconds	46
10.5 Application Specific Platform	47
10.5.1 Detailed Description	47
10.5.2 Function Documentation	48
10.5.2.1 CustomAssertFailed(uint8_t *file, uint32_t line)	48
10.5.2.2 CustomDefaultHandler()	49
10.5.2.3 ExecPlatform()	49
10.5.2.4 HAL_GPIO_EXTI_Callback(uint16_t GPIO_P)	49
10.5.2.5 HAL_I2C_ErrorCallback(I2C_HandleTypeDef *hi2c)	50
10.5.2.6 HAL_I2C_MasterTxCpltCallback(I2C_HandleTypeDef *hi2c)	50
10.5.2.7 HAL_I2C_SlaveTxCpltCallback(I2C_HandleTypeDef *hi2c)	50
10.5.2.8 HAL_SPI_ErrorCallback(SPI_HandleTypeDef *hspi)	51
10.5.2.9 HAL_SPI_TxRxCpltCallback(SPI_HandleTypeDef *hspi)	51
10.5.2.10 HAL_UART_ErrorCallback(UART_HandleTypeDef *huart)	51
10.5.2.11 HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)	52
10.5.2.12 HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)	52
10.5.2.13 InitPlatform()	52
10.5.3 Variable Documentation	53
10.5.3.1 debugger	53
10.6 Abstract Classes	54
10.6.1 Detailed Description	54
10.7 Helper classes	55
10.7.1 Detailed Description	55
10.7.2 Function Documentation	55
10.7.2.1 operator delete(void *ptr)	55

CONTENTS vii

10.7.2.2 operator delete[](void *ptr)	56
10.7.2.3 operator new(std::size_t size)	56
10.7.2.4 operator new[](std::size_t size)	56
10.8 CMSIS	57
10.8.1 Detailed Description	57
10.9 Stm32f7xx_system	58
10.9.1 Detailed Description	58
10.10STM32F7xx_System_Private_Includes	59
10.10.1 Detailed Description	59
10.10.2 Macro Definition Documentation	59
10.10.2.1 HSE_VALUE	59
10.10.2.2 HSI_VALUE	59
10.11STM32F7xx_System_Private_TypesDefinitions	60
10.12STM32F7xx_System_Private_Defines	61
10.12STM32F7xx_System_Private_Defines	61 61
	61
10.12.1 Detailed Description	61 61
10.12.1 Detailed Description	61 61
10.12.1 Detailed Description	61 61
10.12.1 Detailed Description  10.12.2 Macro Definition Documentation  10.12.2.1 VECT_TAB_OFFSET  10.13STM32F7xx_System_Private_Macros	61 61 61 62
10.12.1 Detailed Description  10.12.2 Macro Definition Documentation  10.12.2.1 VECT_TAB_OFFSET  10.13STM32F7xx_System_Private_Macros  10.14STM32F7xx_System_Private_Variables	61 61 62 63
10.12.1 Detailed Description  10.12.2 Macro Definition Documentation  10.12.2.1 VECT_TAB_OFFSET  10.13STM32F7xx_System_Private_Macros  10.14STM32F7xx_System_Private_Variables  10.14.1 Detailed Description	61 61 61 62 63
10.12.1 Detailed Description  10.12.2 Macro Definition Documentation  10.12.2.1 VECT_TAB_OFFSET  10.13STM32F7xx_System_Private_Macros  10.14STM32F7xx_System_Private_Variables  10.14.1 Detailed Description  10.15STM32F7xx_System_Private_FunctionPrototypes	61 61 62 63 64
10.12.1 Detailed Description  10.12.2 Macro Definition Documentation  10.12.2.1 VECT_TAB_OFFSET  10.13STM32F7xx_System_Private_Macros  10.14STM32F7xx_System_Private_Variables  10.14.1 Detailed Description  10.15STM32F7xx_System_Private_FunctionPrototypes  10.16STM32F7xx_System_Private_Functions	61 61 62 63 63 64 65
10.12.1 Detailed Description  10.12.2 Macro Definition Documentation  10.12.2.1 VECT_TAB_OFFSET  10.13STM32F7xx_System_Private_Macros  10.14STM32F7xx_System_Private_Variables  10.14.1 Detailed Description  10.15STM32F7xx_System_Private_FunctionPrototypes  10.16STM32F7xx_System_Private_Functions  10.16.1 Detailed Description	61 61 62 63 63 64 65

viii CONTENTS

11	Nam	espace	Documer	ntation	67
	11.1	murasa	ıki Names	pace Reference	67
		11.1.1	Detailed	Description	68
		11.1.2	Function	Documentation	68
			11.1.2.1	AddSyslogFacilityToMask(murasaki::SyslogFacility facility)	68
			11.1.2.2	AllowedSyslogOut(murasaki::SyslogFacility facility, murasaki::SyslogSeverity severity)	68
			11.1.2.3	RemoveSyslogFacilityFromMask(murasaki::SyslogFacility facility)	69
			11.1.2.4	SetSyslogFacilityMask(uint32_t mask)	69
			11.1.2.5	SetSyslogSererityThreshold(murasaki::SyslogSeverity severity)	69
		11.1.3	Variable	Documentation	69
			11.1.3.1	platform	69
12	Clas	s Docui	mentation		71
	12.1	murasa	aki::Adau1	361 Class Reference	71
		12.1.1	Construc	tor & Destructor Documentation	72
			12.1.1.1	Adau1361(unsigned int fs, murasaki::I2CMasterStrategy *controler, unsigned int i2c_device_addr)	72
		12.1.2	Member	Function Documentation	72
			12.1.2.1	configure_board(void)=0	72
			12.1.2.2	configure_pll(void)=0	73
			12.1.2.3	send_command(const uint8_t command[], int size)	73
			12.1.2.4	send_command_table(const uint8_t table[][3], int rows)	73
			12.1.2.5	set_aux_input_gain(float left_gain, float right_gain, bool mute=false)	73
			12.1.2.6	set_hp_output_gain(float left_gain, float right_gain, bool mute=false)	74
			12.1.2.7	set_line_input_gain(float left_gain, float right_gain, bool mute=false)	74
			12.1.2.8	set_line_output_gain(float left_gain, float right_gain, bool mute=false)	74
			12.1.2.9	start(void)	75
			12.1.2.10	wait_pll_lock(void)	75
	12.2	murasa	aki::AudioC	CodecStrategy Class Reference	75
		12.2.1	Detailed	Description	76
		12.2.2	Construc	tor & Destructor Documentation	76

CONTENTS

	12.2.2.1 AudioCodecStrategy(unsigned int fs)	76
12.2.	3 Member Function Documentation	76
	12.2.3.1 set_aux_input_gain(float left_gain, float right_gain, bool mute=false)	76
	12.2.3.2 set_hp_output_gain(float left_gain, float right_gain, bool mute=false)	76
	12.2.3.3 set_line_input_gain(float left_gain, float right_gain, bool mute=false)	77
	12.2.3.4 set_line_output_gain(float left_gain, float right_gain, bool mute=false)	77
	12.2.3.5 set_mic_input_gain(float left_gain, float right_gain, bool mute=false)	77
	12.2.3.6 start(void)=0	78
12.3 mura	saki::BitIn Class Reference	78
12.3.	1 Detailed Description	79
12.3.	2 Constructor & Destructor Documentation	79
	12.3.2.1 BitIn(GPIO_TypeDef *port, uint16_t pin)	79
12.3.	3 Member Function Documentation	79
	12.3.3.1 Get(void)	79
	12.3.3.2 GetPeripheralHandle()	80
12.4 mura	saki::BitInStrategy Class Reference	80
12.4.	1 Detailed Description	81
12.4.	2 Member Function Documentation	81
	12.4.2.1 Get(void)=0	81
12.5 mura	saki::BitOut Class Reference	81
12.5.	1 Detailed Description	82
12.5.	2 Constructor & Destructor Documentation	82
	12.5.2.1 BitOut(GPIO_TypeDef *port, uint16_t pin)	82
12.5.	3 Member Function Documentation	83
	12.5.3.1 Get(void)	83
	12.5.3.2 GetPeripheralHandle()	83
	12.5.3.3 Set(unsigned int state=1)	83
12.6 mura	saki::BitOutStrategy Class Reference	83
12.6.	1 Detailed Description	84
12.6.	2 Member Function Documentation	84

CONTENTS

12.6.2.1 Get(void)=0	84
12.6.2.2 Set(unsigned int state=1)=0	85
12.7 murasaki::CriticalSection Class Reference	85
12.7.1 Detailed Description	85
12.7.2 Member Function Documentation	85
12.7.2.1 Enter()	85
12.7.2.2 Leave()	86
12.8 murasaki::Debugger Class Reference	86
12.8.1 Detailed Description	87
12.8.2 Constructor & Destructor Documentation	87
12.8.2.1 Debugger(LoggerStrategy *logger)	87
12.8.3 Member Function Documentation	87
12.8.3.1 AutoRePrint()	87
12.8.3.2 GetchFromTask()	87
12.8.3.3 Printf(const char *fmt,)	88
12.8.3.4 RePrint()	88
12.8.4 Member Data Documentation	88
12.8.4.1 facility_mask	88
12.8.4.2 line	88
12.8.4.3 severity	89
12.9 murasaki::DebuggerFifo Class Reference	89
12.9.1 Detailed Description	90
12.9.2 Constructor & Destructor Documentation	90
12.9.2.1 DebuggerFifo(unsigned int buffer_size)	90
12.9.3 Member Function Documentation	90
12.9.3.1 Get(uint8_t data[], unsigned int size)	90
12.9.3.2 SetPostMortem()	91
12.10murasaki::DebuggerUart Class Reference	91
12.10.1 Detailed Description	92
12.10.2 Constructor & Destructor Documentation	92

CONTENTS xi

12.10.2.1 DebuggerUart(UART_HandleTypeDef *uart)	92
12.10.3 Member Function Documentation	93
12.10.3.1 HandleError(void *const ptr)	93
12.10.3.2 Receive(uint8_t ∗data, unsigned int count, unsigned int ∗transfered_count, Uart ← Timeout uart_timeout, WaitMilliSeconds timeout_ms)	93
12.10.3.3 ReceiveCompleteCallback(void *const ptr)	94
12.10.3.4 SetHardwareFlowControl(UartHardwareFlowControl control)	94
12.10.3.5 SetSpeed(unsigned int baud_rate)	94
12.10.3.6 Transmit(const uint8_t *data, unsigned int size, WaitMilliSeconds timeout_ms) .	95
12.10.3.7 TransmitCompleteCallback(void *const ptr)	95
12.11 murasaki::FifoStrategy Class Reference	96
12.11.1 Detailed Description	96
12.11.2 Constructor & Destructor Documentation	96
12.11.2.1 FifoStrategy(unsigned int buffer_size)	96
12.11.3 Member Function Documentation	97
12.11.3.1 Get(uint8_t data[], unsigned int size)	97
12.11.3.2 Put(uint8_t const data[], unsigned int size)	97
12.12murasaki::GPIO_type Struct Reference	97
12.12.1 Detailed Description	98
12.13murasaki::I2cMaster Class Reference	98
12.13.1 Detailed Description	99
12.13.2 Constructor & Destructor Documentation	100
12.13.2.1 I2cMaster(I2C_HandleTypeDef *i2c_handle)	100
12.13.3 Member Function Documentation	101
12.13.3.1 HandleError(void *ptr)	101
12.13.3.2 Receive(uint addrs, uint8_t *rx_data, unsigned int rx_size, uint *transfered_count, WaitMilliSeconds timeout_ms)	101
12.13.3.3 ReceiveCompleteCallback(void *ptr)	102
12.13.3.4 Transmit(uint addrs, const uint8_t *tx_data, unsigned int tx_size, uint *transfered_count, WaitMilliSeconds timeout_ms)	102
12.13.3.5 TransmitCompleteCallback(void *ptr)	103

xii CONTENTS

12.13.3.6 TransmitThenReceive(uint addrs, const uint8_t *tx_data, unsigned int tx_size, uint8_t *rx_data, unsigned int rx_size, uint *tx_transfered_count, uint *rx_← transfered_count, WaitMilliSeconds timeout_ms)	103
12.14murasaki::I2CMasterStrategy Class Reference	104
12.14.1 Detailed Description	105
12.14.2 Member Function Documentation	105
12.14.2.1 HandleError(void *ptr)=0	105
12.14.2.2 Receive(uint addrs, uint8_t *rx_data, unsigned int rx_size, uint *transfered_← count=nullptr, WaitMilliSeconds timeout_ms=murasaki::kwmsIndefinitely)=0	106
12.14.2.3 ReceiveCompleteCallback(void *ptr)=0	106
12.14.2.4 Transmit(uint addrs, const uint8_t *tx_data, unsigned int tx_size, uint *transfered_count=nullptr, WaitMilliSeconds timeout_ms=murasaki::kwms↔ Indefinitely)=0	107
12.14.2.5 TransmitCompleteCallback(void *ptr)=0	107
12.14.2.6 TransmitThenReceive(uint addrs, const uint8_t *tx_data, unsigned int tx_size, uint8_t *rx_data, unsigned int rx_size, uint *tx_transfered_count=nullptr, uint *rx_transfered_count=nullptr, WaitMilliSeconds timeout_ms=murasaki::kwms← Indefinitely)=0	107
12.15murasaki::l2cSlave Class Reference	
12.15.1 Detailed Description	109
12.15.2 Member Function Documentation	110
12.15.2.1 HandleError(void *ptr)	110
12.15.2.2 Receive(uint8_t *rx_data, unsigned int rx_size, uint *transfered_count, Wait← MilliSeconds timeout_ms)	110
12.15.2.3 ReceiveCompleteCallback(void *ptr)	111
12.15.2.4 Transmit(const uint8_t *tx_data, unsigned int tx_size, uint *transfered_count, WaitMilliSeconds timeout_ms)	111
12.15.2.5 TransmitCompleteCallback(void *ptr)	112
12.16murasaki::l2cSlaveStrategy Class Reference	113
12.16.1 Detailed Description	114
12.16.2 Member Function Documentation	114
12.16.2.1 HandleError(void *ptr)=0	114
12.16.2.2 Receive(uint8_t *rx_data, unsigned int rx_size, uint *transfered_count=nullptr, murasaki::WaitMilliSeconds timeout_ms=murasaki::kwmsIndefinitely)=0	114
12.16.2.3 ReceiveCompleteCallback(void *ptr)=0	115

CONTENTS xiii

12.16.2.4 Iransmit(const uint8_t *tx_data, unsigned int tx_size, uint *transfered↔ _count=nullptr, murasaki::WaitMilliSeconds timeout_ms=murasaki::kwms↔	445
Indefinitely)=0	115
12.16.2.5 TransmitCompleteCallback(void *ptr)=0	115
12.17murasaki::LoggerStrategy Class Reference	116
12.17.1 Detailed Description	116
12.17.2 Constructor & Destructor Documentation	117
12.17.2.1 ~LoggerStrategy()	117
12.17.3 Member Function Documentation	117
12.17.3.1 DoPostMortem(void *debugger_fifo)	117
12.17.3.2 getCharacter()=0	117
12.17.3.3 putMessage(char message[], unsigned int size)=0	117
12.18murasaki::LoggingHelpers Struct Reference	118
12.19murasaki::PeripheralStrategy Class Reference	118
12.19.1 Detailed Description	119
12.20murasaki::Platform Struct Reference	119
12.20.1 Detailed Description	120
12.21 murasaki::SpiMaster Class Reference	120
12.21.1 Detailed Description	121
12.21.2 Constructor & Destructor Documentation	121
12.21.2.1 SpiMaster(SPI_HandleTypeDef *spi_handle)	121
12.21.3 Member Function Documentation	122
12.21.3.1 HandleError(void *ptr)	122
12.21.3.2 TransmitAndReceive(murasaki::SpiSlaveSpecifierStrategy *spi_spec, const uint8_t *tx_data, uint8_t *rx_data, unsigned int size, murasaki::WaitMilliSeconds timeout_ms=murasaki::kwmsIndefinitely)	122
12.21.3.3 TransmitAndReceiveCompleteCallback(void *ptr)	123
12.22murasaki::SpiMasterStrategy Class Reference	123
12.22.1 Detailed Description	124
12.22.2 Member Function Documentation	124
12.22.2.1 HandleError(void *ptr)=0	124

XIV

12.22.2.2 TransmitAndReceive(murasaki::SpiSlaveSpecifierStrategy *spi_spec, const uint8_t *tx_data, uint8_t *rx_data, unsigned int size, murasaki::WaitMilliSeconds	405
timeout_ms=murasaki::kwmsIndefinitely)=0	125
12.22.2.3 TransmitAndReceiveCompleteCallback(void *ptr)=0	125
12.23murasaki::SpiSlave Class Reference	126
12.23.1 Detailed Description	127
12.23.2 Constructor & Destructor Documentation	127
12.23.2.1 SpiSlave(SPI_HandleTypeDef *spi_handle)	127
12.23.3 Member Function Documentation	127
12.23.3.1 HandleError(void *ptr)	128
12.23.3.2 TransmitAndReceive(const_uint8_t *tx_data, uint8_t *rx_data, unsigned int size, unsigned int *transfered_count, murasaki::WaitMilliSeconds timeout_← ms=murasaki::kwmsIndefinitely)	128
12.23.3.3 TransmitAndReceiveCompleteCallback(void *ptr)	129
12.24murasaki::SpiSlaveSpecifier Class Reference	129
12.24.1 Detailed Description	130
12.24.2 Constructor & Destructor Documentation	130
12.24.2.1 SpiSlaveSpecifier(murasaki::SpiClockPolarity pol, murasaki::SpiClockPhase pha,::GPIO_TypeDef *port, uint16_t pin)	130
pha,::GPIO_TypeDef *port, uint16_t pin)	131
pha,::GPIO_TypeDef *port, uint16_t pin)	131 131
pha,::GPIO_TypeDef *port, uint16_t pin)  12.24.2.2 SpiSlaveSpecifier(unsigned int pol, unsigned int pha,::GPIO_TypeDef *const port, uint16_t pin)  12.24.3 Member Function Documentation	131 131 131
pha,::GPIO_TypeDef *port, uint16_t pin)  12.24.2.2 SpiSlaveSpecifier(unsigned int pol, unsigned int pha,::GPIO_TypeDef *const port, uint16_t pin)  12.24.3 Member Function Documentation  12.24.3.1 AssertCs()	131 131 131 131
pha,::GPIO_TypeDef *port, uint16_t pin)  12.24.2.2 SpiSlaveSpecifier(unsigned int pol, unsigned int pha,::GPIO_TypeDef *const port, uint16_t pin)  12.24.3 Member Function Documentation  12.24.3.1 AssertCs()  12.24.3.2 DeassertCs()	131 131 131 131 132
pha,::GPIO_TypeDef *port, uint16_t pin)  12.24.2.2 SpiSlaveSpecifier(unsigned int pol, unsigned int pha,::GPIO_TypeDef *const port, uint16_t pin)  12.24.3 Member Function Documentation  12.24.3.1 AssertCs()  12.24.3.2 DeassertCs()  12.25murasaki::SpiSlaveSpecifierStrategy Class Reference	131 131 131 131 132
pha,::GPIO_TypeDef *port, uint16_t pin)  12.24.2.2 SpiSlaveSpecifier(unsigned int pol, unsigned int pha,::GPIO_TypeDef *const port, uint16_t pin)  12.24.3 Member Function Documentation  12.24.3.1 AssertCs()  12.24.3.2 DeassertCs()  12.25murasaki::SpiSlaveSpecifierStrategy Class Reference  12.25.1 Detailed Description	131 131 131 131 132 132
pha,::GPIO_TypeDef *port, uint16_t pin)  12.24.2.2 SpiSlaveSpecifier(unsigned int pol, unsigned int pha,::GPIO_TypeDef *const port, uint16_t pin)  12.24.3 Member Function Documentation  12.24.3.1 AssertCs()  12.24.3.2 DeassertCs()  12.25murasaki::SpiSlaveSpecifierStrategy Class Reference  12.25.1 Detailed Description  12.25.2 Constructor & Destructor Documentation  12.25.2.1 SpiSlaveSpecifierStrategy(murasaki::SpiClockPolarity pol, murasaki::SpiClock	131 131 131 132 132 132
pha,::GPIO_TypeDef *port, uint16_t pin)  12.24.2.2 SpiSlaveSpecifier(unsigned int pol, unsigned int pha,::GPIO_TypeDef *const port, uint16_t pin)  12.24.3 Member Function Documentation  12.24.3.1 AssertCs()  12.24.3.2 DeassertCs()  12.25murasaki::SpiSlaveSpecifierStrategy Class Reference  12.25.1 Detailed Description  12.25.2 Constructor & Destructor Documentation  12.25.2.1 SpiSlaveSpecifierStrategy(murasaki::SpiClockPolarity pol, murasaki::SpiClock→Phase pha)	131 131 131 132 132 132 133
pha,::GPIO_TypeDef *port, uint16_t pin)  12.24.2.2 SpiSlaveSpecifier(unsigned int pol, unsigned int pha,::GPIO_TypeDef *const port, uint16_t pin)  12.24.3 Member Function Documentation  12.24.3.1 AssertCs()  12.24.3.2 DeassertCs()  12.25murasaki::SpiSlaveSpecifierStrategy Class Reference  12.25.1 Detailed Description  12.25.2 Constructor & Destructor Documentation  12.25.2.1 SpiSlaveSpecifierStrategy(murasaki::SpiClockPolarity pol, murasaki::SpiClock→Phase pha)  12.25.2.2 SpiSlaveSpecifierStrategy(unsigned int pol, unsigned int pha)	131 131 131 132 132 132 133 133

CONTENTS xv

12.25.3.3 GetCpha()	133
12.25.3.4 GetCpol()	134
12.26murasaki::SpiSlaveStrategy Class Reference	134
12.26.1 Detailed Description	135
12.26.2 Member Function Documentation	135
12.26.2.1 HandleError(void *ptr)=0	135
12.26.2.2 TransmitAndReceive(const uint8_t *tx_data, uint8_t *rx_data, unsigned int size, unsigned int *transfered_count=nullptr, murasaki::WaitMilliSeconds timeout_ ms=murasaki::kwmsIndefinitely)=0	135
12.26.2.3 TransmitAndReceiveCompleteCallback(void *ptr)=0	136
12.27murasaki::Synchronizer Class Reference	136
12.27.1 Detailed Description	136
12.27.2 Member Function Documentation	136
12.27.2.1 Release()	136
12.27.2.2 Wait(WaitMilliSeconds timeout_ms=kwmsIndefinitely)	136
12.28murasaki::Task Class Reference	137
12.28.1 Detailed Description	138
12.28.2 Constructor & Destructor Documentation	138
12.28.2.1 Task(const char *task_name, unsigned short stack_depth, UBaseType_t task_ priority, const void *task_parameter, void(*task_body_func)(const void *))	138
12.28.3 Member Function Documentation	139
12.28.3.1 TaskBody(const void *ptr)	139
12.29murasaki::TaskStrategy Class Reference	139
12.29.1 Detailed Description	140
12.29.2 Constructor & Destructor Documentation	140
12.29.2.1 TaskStrategy(const char *task_name, unsigned short stack_depth, UBaseType ← _t task_priority, const void *task_parameter)	140
12.29.3 Member Function Documentation	140
12.29.3.1 GetName()	140
12.29.3.2 Launch(void *ptr)	140
12.29.3.3 Start()	141
12.29.3.4 TaskBody(const void *ptr)=0	141

xvi CONTENTS

12.30murasaki::Uart Class Reference	141
12.30.1 Detailed Description	143
12.30.2 Constructor & Destructor Documentation	143
12.30.2.1 Uart(UART_HandleTypeDef *uart)	143
12.30.3 Member Function Documentation	144
12.30.3.1 HandleError(void *const ptr)	144
12.30.3.2 Receive(uint8_t ∗data, unsigned int count, unsigned int ∗transfered_count, Uart ← Timeout uart_timeout, WaitMilliSeconds timeout_ms)	144
12.30.3.3 ReceiveCompleteCallback(void *const ptr)	145
12.30.3.4 SetHardwareFlowControl(UartHardwareFlowControl control)	145
12.30.3.5 SetSpeed(unsigned int baud_rate)	146
12.30.3.6 Transmit(const uint8_t *data, unsigned int size, WaitMilliSeconds timeout_ms) .	146
12.30.3.7 TransmitCompleteCallback(void *const ptr)	146
12.31 murasaki::UartLogger Class Reference	147
12.31.1 Detailed Description	148
12.31.2 Constructor & Destructor Documentation	148
12.31.2.1 UartLogger(UartStrategy *uart)	148
12.31.3 Member Function Documentation	148
12.31.3.1 DoPostMortem(void *debugger_fifo)	149
12.31.3.2 getCharacter()	149
12.31.3.3 putMessage(char message[], unsigned int size)	149
12.32murasaki::UartStrategy Class Reference	149
12.32.1 Detailed Description	151
12.32.2 Member Function Documentation	151
12.32.2.1 HandleError(void *ptr)=0	151
12.32.2.2 Receive(uint8_t *data, unsigned int size, unsigned int *transfered_count=nullptr, UartTimeout uart_timeout=murasaki::kutNoldleTimeout, WaitMilliSeconds timeout_ms=murasaki::kwmsIndefinitely)=0	151
12.32.2.3 ReceiveCompleteCallback(void *ptr)=0	152
12.32.2.4 SetHardwareFlowControl(UartHardwareFlowControl control)	152
12.32.2.5 SetSpeed(unsigned int speed)	152
12.32.2.6 Transmit(const uint8_t *data, unsigned int size, WaitMilliSeconds timeout_← ms=murasaki::kwmsIndefinitely)=0	152
12.32.2.7 TransmitCompleteCallback(void *ptr)=0	153

CONTENTS xvii

13 File	Docum	entation		155
13.1	1 /home/	/takemasa/	murasaki_samples/nucleo-f746-sample/Inc/main.h File Reference	155
	13.1.1	Detailed I	Description	156
	13.1.2	Function	Documentation	156
		13.1.2.1	Error_Handler(void)	156
13.2	2 /home/	/takemasa/	murasaki_samples/nucleo-f746-sample/Inc/murasaki_platform.hpp File Reference	157
	13.2.1	Detailed I	Description	157
13.3	3 /home/	/takemasa/	murasaki_samples/nucleo-f746-sample/Inc/platform_config.hpp File Reference	158
	13.3.1	Detailed I	Description	159
	13.3.2	Macro De	finition Documentation	159
		13.3.2.1	MURASAKI_CONFIG_NOSYSLOG	159
13.4	4 /home/	/takemasa/	murasaki_samples/nucleo-f746-sample/Inc/platform_defs.hpp File Reference	159
	13.4.1	Detailed I	Description	160
13.5	5 /home/	/takemasa/	murasaki_samples/nucleo-f746-sample/Inc/stm32f7xx_hal_conf.h File Reference	160
	13.5.1	Detailed I	Description	161
	13.5.2	Macro De	finition Documentation	162
		13.5.2.1	assert_param	162
		13.5.2.2	EXTERNAL_CLOCK_VALUE	162
		13.5.2.3	HSE_STARTUP_TIMEOUT	162
		13.5.2.4	HSE_VALUE	162
		13.5.2.5	HSI_VALUE	163
		13.5.2.6	LSE_STARTUP_TIMEOUT	163
		13.5.2.7	LSE_VALUE	163
		13.5.2.8	LSI_VALUE	163
		13.5.2.9	PHY_AUTONEGO_COMPLETE	163
		13.5.2.10	PHY_AUTONEGOTIATION	163
		13.5.2.11	PHY_BCR	163
		13.5.2.12	PHY_BSR	163
		13.5.2.13	PHY_DUPLEX_STATUS	163
		13.5.2.14	PHY_FULLDUPLEX_100M	164

xviii CONTENTS

13.5.2.15 PHY_FULLDUPLEX_10M	164
13.5.2.16 PHY_HALFDUPLEX_100M	164
13.5.2.17 PHY_HALFDUPLEX_10M	164
13.5.2.18 PHY_ISOLATE	164
13.5.2.19 PHY_JABBER_DETECTION	164
13.5.2.20 PHY_LINKED_STATUS	164
13.5.2.21 PHY_LOOPBACK	164
13.5.2.22 PHY_POWERDOWN	164
13.5.2.23 PHY_RESET	164
13.5.2.24 PHY_RESTART_AUTONEGOTIATION	165
13.5.2.25 PHY_SPEED_STATUS	165
13.5.2.26 PHY_SR	165
13.5.2.27 TICK_INT_PRIORITY	165
13.5.2.28 VDD_VALUE	165
13.5.3 Function Documentation	165
13.5.3.1 assert_failed(uint8_t *file, uint32_t line)	165
13.6 /home/takemasa/murasaki_samples/nucleo-f746-sample/Inc/stm32f7xx_it.h File Reference	166
13.6.1 Detailed Description	166
13.7 /home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc-tp/adau1361.hpp File Reference	167
13.7.1 Detailed Description	167
13.8 /home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/audiocodecstrategy.hpp File Reference	168
13.8.1 Detailed Description	168
13.9 /home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/bitin.hpp File Reference	169
13.9.1 Detailed Description	170
13.10/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/bitinstrategy.hpp File Reference	171
13.10.1 Detailed Description	172
13.11/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/bitout.hpp File Reference	173
13.11.1 Detailed Description	174

CONTENTS xix

13.12/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/bitoutstrategy.hpp File Reference	175
13.12.1 Detailed Description	176
13.13/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/criticalsection.hpp File Reference	177
13.13.1 Detailed Description	177
13.14/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/debugger.hpp File Reference	178
13.14.1 Detailed Description	179
13.15/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/debuggerfifo.hpp File Reference	179
13.15.1 Detailed Description	180
13.16/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/debuggeruart.hpp File Reference	181
13.16.1 Detailed Description	182
13.17/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/fifostrategy.hpp File Reference	<del>9</del> 183
13.17.1 Detailed Description	184
13.18/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/i2cmaster.hpp File Reference	184
13.18.1 Detailed Description	185
13.19/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/i2cmasterstrategy.hpp File Reference	186
13.19.1 Detailed Description	187
13.20/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/i2cslave.hpp File Reference	188
13.20.1 Detailed Description	189
13.21/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/i2cslavestrategy.hpp File Reference	190
13.21.1 Detailed Description	191
13.22/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/loggerstrategy.hpp File Reference	192
13.22.1 Detailed Description	193
13.23/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/murasaki.hpp File Reference	194
13.23.1 Detailed Description	195
13.24/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/murasaki_0_intro.hpp File Reference	195
13.24.1 Detailed Description	195

CONTENTS

13.25/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/murasaki_1_env.hpp File Reference	195
13.25.1 Detailed Description	195
13.26/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/murasaki_2_ug.hpp File Reference	195
13.26.1 Detailed Description	195
13.27/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/murasaki_3_pg.hpp File Reference	196
13.27.1 Detailed Description	196
13.28/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/murasaki_4_mod.hpp File Reference	196
13.28.1 Detailed Description	196
13.29/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/murasaki_assert.hpp File Reference	196
13.29.1 Detailed Description	197
13.30/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/murasaki_config.hpp File Reference	198
13.30.1 Detailed Description	199
13.31/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/murasaki_defs.hpp File Reference	199
13.31.1 Detailed Description	200
13.32/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/murasaki_syslog.hpp File Reference	200
13.32.1 Detailed Description	201
13.33/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/peripheralstrategy.hpp File Reference	202
13.33.1 Detailed Description	203
13.34/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/spimaster.hpp File Reference	203
13.34.1 Detailed Description	204
13.35/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/spimasterstrategy.hpp File Reference	205
13.35.1 Detailed Description	206
13.36/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/spislave.hpp File Reference	207
13.36.1 Detailed Description	208

CONTENTS xxi

13.37/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/spislavespecifier.hpp File Reference	209
13.37.1 Detailed Description	210
13.38/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/spislavespecifierstrategy.hpp	
File Reference	211
13.38.1 Detailed Description	212
13.39/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/spislavestrategy.hpp File Reference	213
13.39.1 Detailed Description	214
13.40/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/synchronizer.hpp File Refer-	
ence	215
13.40.1 Detailed Description	216
13.41/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/task.hpp File Reference	216
13.41.1 Detailed Description	217
13.42/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/taskstrategy.hpp File Refer-	
ence	218
13.42.1 Detailed Description	219
13.43/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/uart.hpp File Reference	219
13.43.1 Detailed Description	220
13.44/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/uartlogger.hpp File Reference	221
13.44.1 Detailed Description	222
13.45/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/uartstrategy.hpp File Reference	223
13.45.1 Detailed Description	224
13.46/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Src/allocators.cpp File Reference	224
13.46.1 Detailed Description	225
13.47/home/takemasa/murasaki_samples/nucleo-f746-sample/Src/main.c File Reference	225
13.47.1 Detailed Description	226
13.47.2 Function Documentation	226
13.47.2.1 assert_failed(uint8_t *file, uint32_t line)	226
13.47.2.2 Error_Handler(void)	227
13.47.2.3 HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)	227
13.47.2.4 main(void)	227

xxii CONTENTS

13.47.2.5 StartDefaultTask(void const *argument)	227
13.47.2.6 SystemClock_Config(void)	228
13.47.3 Variable Documentation	228
13.47.3.1 hdma_spi1_tx	228
13.48/home/takemasa/murasaki_samples/nucleo-f746-sample/Src/murasaki_platform.cpp File Reference	229
13.48.1 Detailed Description	230
13.48.2 Function Documentation	230
13.48.2.1 HAL_I2C_MasterRxCpltCallback(I2C_HandleTypeDef *hi2c)	230
13.48.2.2 HAL_I2C_SlaveRxCpltCallback(I2C_HandleTypeDef *hi2c)	230
13.49/home/takemasa/murasaki_samples/nucleo-f746-sample/Src/stm32f7xx_it.c File Reference	231
13.49.1 Detailed Description	231
13.49.2 Variable Documentation	232
13.49.2.1 hdma_spi1_tx	232
13.50/home/takemasa/murasaki_samples/nucleo-f746-sample/Src/system_stm32f7xx.c File Reference .	232
13.50.1 Detailed Description	233
Index	235

# **Chapter 1**

# **Preface**

Murasaki, is a class library on the STM32Cube HAL and FreeRTOS.

By using Murasaki, you can program STM32 series quickly and easily.

Murasaki has following design philosophies:

- · Simplified IO
- · Preemptive multi-task
- Blocking IO
- · Thread safe IO
- · Versatile printf() logger
- · Guard by assertion
- · System Logging
- Configurable

# 1.1 Simplified IO

The IO function is packaged by class types. For example, The murasaki:: Uart class can receive a UART handle

```
murasaki::AbstractUart * uart3 = new murasaki::Uart( &huart3 );
```

Where huart3 is a UART port 3 handle generated by the CubeMX.

The STM32Cube HAL is quite rich and flexible. On the other hand, it is quite huge and complex. The classes in Murasaki simplifies it by letting flexibility beside. For example, the murasaki::Uart class can support only the DMA transfer. The interrupt-based transfer is not supported. By giving up the flexibility, programming with Murasaki is easier than using HAL directly.

2 Preface

## 1.2 Preemptive multi-task

The Murasaki class library is buit on FreeRTOS's preemptive configuration. As a result, Murasaki is automatically aware with preemptive multi-task.

That means, Murasaki's classes don't use polling to wait for any event. Then, a task can do some job while other tasks are waiting for some event.

The multi-task programming helps to divide a bigger program to sub-units. This is a good way to develop a large program easier. And the more important point, it is easier to maintain.

### 1.3 Blocking IO

The blocking IO is one of the most important features of Murasaki.

The peripheral wrapping class like murasaki::Uart provides a set of member functions to do the data transmission/receiving. Such the member functions are programmed as "blocking" IO.

The blocking IO function doesn't return until each IO function finished completely. For example, if you transmit 10bytes through the UART, the IO member function transmits the 10bytes data, and then, return.

Note: Sometimes, the "completion" means the end of the DMA transfer session, rather than the true transmission of the last byte. In this case, system generates a completon interrupt while the data is still in FIFO of the peripheral. This is a hardware issue.

To provide the blocking IO, some member functions are restricted to use only in the task context.

#### 1.4 Thread safe IO

The blocking IO and the preemptive multi-task provide easier programming. In the other hand, there is a possibility that two different task accesses one peripheral simultaneously. This kind of access messes the peripheral's behavior.

To prevent this condition, each peripheral wrapping class has exclusive access mechanism by mutex.

By this mechanism, if two tasks try to transmit though one peripheral, one task is kept waiting until the other finished to transmit.

# 1.5 Versatile printf() logger

Logging or "printf debug" is a strong tool in the embedded system development.

Murasaki has three levels of the printf debugging mechanism. One is the murasaki::debugger->Printf(), the second is MURASAKI\_ASSERT macro. In addition to these two, MURASAKI\_SYSLOG macro is avairable.

The murasaki::debugger->Printf() is flexible output mechanism which has several good features :

- · printf() compatible parameters.
- · Task/interrupt bi-context operation
- None-blocking logging by internal buffer.
- · User configurable output port

These features allow a programmer to do the printf() debug not only in the task context but also in the interrupt context.

1.6 Guard by assertion 3

## 1.6 Guard by assertion

In addition to the murasaki::debugger->Printf(), programmer can use MURASAKI\_ASSERT macro. This allows easy assertion and logging. This macro uses the murasaki::debugger->Printf() internally.

This assertion is used inside Murasaki class library. As a result, the wrong context, wrong parameter, etc will be reported to the debugger output.

# 1.7 System Logging

MURASAKI\_SYSLOG provides the message output based on the level and filtering. This mechanism is intended to help the Murasaki library development. But also application can use this mechanism.

## 1.8 Configurable

Murasaki is configurable from the two point of view.

First, Musaraki's modules enable only when the relevant peripheral is generated by CubeMX. This allows you set the CubeMX to generate only the used peripheral's source code. Such the setting makes total source code smaller. In the other hand, all unused drivers are invisible. For example, if you don't enable the I2C pins on CubeMX, Murasaki cannot see such the module.

Murasaki can adopt such the situation. The source code of Murasaki relevant to the peripheral which is not generated, will be disabled by ifdef control.

The Second part of the configurable characteristics is Murasaki itself. The programmer can customize the Murasaki for example, task stack size.

4 Preface

# **Chapter 2**

# **Target and Environment**

Murasaki library was originally developed with following environment:

```
Nucleo F746ZG (STM32F746ZG)
STM32CubeMX 5.0
SW4STM32 1.16.0.201807130628 (with eclipse 4.6.3)
Ubuntu 16.04.03 (64bit)
```

And then, confirmed portability with following boards:

```
Nucleo F746ZG ( STM32F746ZG : Cortex-M7 )
Nucleo F722ZE ( STM32F722ZE : Cortex-M7 ),
Nucleo F303K8 ( STM32F303K8 : Cortex-M4 )
Nucleo L152RE ( STM32L152RE : Cortex-M3 )
Nucleo F091RC ( STM32F091RC : Cortex-M0 )
```

# **Chapter 3**

# **Usage Introduction**

In this introduction, we see how to use Murasaki class library in the STM32 program.

In this seciton, we see fowling issues:

- · Message output
- · Serial communication
- · Debugging with Murasaki.
- Tasking
- · Other peripheral

For the easy-to-understand description, we assumes several things on the application skeleton which we are going to use Murasaki:

- The application skeleton is generated by CubeMX
- The application skeleton is configured to use FreeRTOS
- · UART3 is configured to work with DMA.

### 3.1 Message output

The Murasaki library has a Printf() like message output mechanism.

This mechanism is easy way to display a message from an embedded microcomputer to the terminal simulator like kermit on a host computer. Murasaki's Printf() is based on the standard C language formating library. So, programmer can output a message as like standard printf().

As usual, let's start from "hello, world".

8 Usage Introduction

In Murasaki manner, the Printf() is not a global function. This is a method of murasaki::Debugger class. The murasaki::debugger variable is a one of two Murasaki's golobal variable. And it provide an easy to use message output.

The end-of-line charater is depend on the terminal. In the above sample, the terminator is . This is for the linux based kermit. Other terminal system may need other end-of-line character.

Because the Printf() works as like standard printf(), you can also use the format string.

```
murasaki::debugger->Printf("count is %d\n\r", count);
```

The Printf() is designed as debugger message output for an embeded realtime system. Thenk this function is:

- · Thread safe
- · Blocking
- · Buffered

In the other word, you can use this function in either task or interrupt handler without bothering the real time process.

#### 3.2 Serial communication

murasaki::Uart is the asynchronous serial communication.

The initial baud rate, parity and data size are defined by CubeMX. So, there is no need to initialize the communication parameter in application program. User can transmit data by just passing its address and size.

Beside of transmit, also Receive() member function exists.

### 3.3 Debugging with Murasaki.

As we saw, Murasaki has simple messaging output for realtime debug.

This feature is typically used as UART serial output, but configurable by programmer.

The murasaki::debugger is the useful variable to output the debugging message. murasaki::debugger->prrntf() has several good feature.

- · Versatile printf() style format string.
- · Can call from both task and interrupt context
- Non blocking

These features helps programmer to display message in the real-time, multi-task application.

In addition to this simple debugging variable, a programmer can use assert\_failure() function of the STM32 HA. The STM32Cube HAL has assert\_failure() to check the parameter on the fly. By default, this function is disabled. To use this function, programmer have to make it enable, and add function to receive the debug information.

To enable the assert\_failuer(), edit the stm32fxx\_hal\_conf.h in the Inc directory. This file is generated by CubeMX. You can find USE\_FULL\_ASERT macro as comment out. By declaring this macro, assert\_failure is enabled.

```
#define USE FULL ASSERT 1
```

And then, you should modify assert failure() in main.c, to call output function.

```
void assert_failed(uint8_t* file, uint32_t line)
{
    CustomAssertFailed(file, line); // debugging stub.
}
```

Finally, you must define the output function.

```
// Hook for the assert_failure() in main.c
void CustomAssertFailed(uint8_t* file, uint32_t line)
{
    murasaki::debugger->Printf("Wrong parameters value: file %s on line %d\r\n",
        file, line);
}
```

Once above programming is done, you can watch the integrity of the HAL parameter by reading the console output.

Above debugging mechanism redirects all HAL assertion, Murasaki assertion and application debug message to the specified logging port. That logging port is able to customize. In the case of the User's Guide, logging is done through the UART port.

Time by time, you may not wnat to connect serial terminal to the board, unless you have problem. That means, when you find problem and connect your serial terminal, the assertion message is already transmitted ( and lost ).

Murasaki can save this problem. By adding following code after creating murasaki::Debugger instance, you can use history functionality.

```
murasaki::debugger->AutoHistory();
```

The murasaki::Debugger::AutoHistory() creates a dedicated task for auto history function. This task watch the input from the logging port. Again, in this User's guide it is UART. Once any character is received from the logging port (terminal), previously transmitted message are sent again. So you can read the last tens of messages.

The auto history is handy, but it blocks all input from the terminal. If you want to have your own console program through the debug port input, do not you the auto history. Alternatively, you can send the previously transmitted message again, by calling murasaki::Debugger::PrintHistory() explicitly.

10 Usage Introduction

## 3.4 Tasking

murasaki::Task is a type of the task of the FreeRTOS.

By using murasaki::Task, a programmer can easily create a task object. This object encapsulate the task of the FreeRTOS.

First of all, you must define a task body function. Any function name is acceptable, Only the return type and parameter type is specified.

Then, create a Task object.

There are several parameter to pass for the constructor. The first parameter is the name of the task in FreeRTOS. The second one is the task stack size. This size is depend on the task body function. The third one is the priority of the new task. This bigger value is the higher priority. The fourth one is the pointer to the task parameter. This parameter is passed to the task function body. And then, the last one is the pointer to the task body function.

Once task object is created, you must call Start() member function to start the task.

```
murasaki::platform.task1->Start();
```

Then, task you can call Start() member function to run.

# 3.5 Other peripheral

This section shows samples of the other peripherals.

- I2C Master
- I2C Slave
- SPI Master
- SPI Slave
- GPIO

3.5 Other peripheral

#### 3.5.1 I2C Master

murasaki::I2cMaster class provides the serial communication

The I2C master is easy to use. To send a message to the slave device, you need to specify the slave address in 7bits, pointer to data and data size in byte.

In addition to the Transmit(), murasaki::I2cMaster class has Receive(), and TransmitThenReceive() member function.

#### 3.5.2 I2C Slave

murasaki::I2cSlave class provides the I2C slave function.

The I2C slave is much easier than master, because it doesn't need to specify the slave address. The I2C slave device address is given by CubeMX.

In addition to the Transmit(), murasaki::I2cSlave class has Receive() member function.

#### 3.5.3 SPI Master

murasaki::SpiMaster is the SPI master class of Murasaki.

This class is more complicated than other peripherals, because of flexibility. The SPI master controller must adapt to the several variation of the SPI communication.

- · CPOL configuration
- · CPHA configuration
- · GPIO port configuration to select a slave

The flexibility to above configurations need special mechanism. In Murasaki, this flexibility is responsibility of the murasaki::SpiSlaveSpecifier class. This class holds these configuration. Then, passed to the master class.

So, you must create a murasaki::SpiSlaveSpecifier class object, at first.

Then, you can pass the SpiSlaveSpecifier class object to the murasaki::SpiMaster::TransmitAndRecieve() function.

12 Usage Introduction

# 3.5.4 SPI Slave

murasaki::SpiSlave class provides the SPI slave functionality.

This class encapsulate the SPI slave function.

# 3.5.5 GPIO

murasaki::BitOut and murasaki::BitIn provides the GPIO functionality

Following is the example of the murasaki::BitOut class.

```
// Toggle LED.
murasaki::platform.led->Toggle();
```

In addition to the Toggle(), BitIn has Set() and Clear() member function.

# **Chapter 4**

# Porting guide

This porting guide introduces murasaki class library porting step by step.

In this guide, user will study the library porting to the STM32 microcomputer system working with STM32Cube HAL.

Followings are the contents of this porting guide:

- Directory Structure
- CubeMX setting
- Configuration
- Task Priority and Stack Size
- Heap memory consideration
- · Platform variable
- · Routing interrupts
- Error handling
- · Summary of the porting

There are some other manuals of murasaki class library :

- Preface
- Usage Introduction
- · Murasaki Class Collection

### 4.1 Directory Structure

Murasaki has four main directory and several user-modifiable files.

This page describes these directories and files.

14 Porting guide

#### 4.1.1 Src directory

Almost files of the Murasaki source code are stored in this directory. Basically, there is no need to edit the files inside this directory, except the development of Murasaki itself. The project setting must refer this directory as the source directory.

#### 4.1.2 Inc directory

This directory contains the include files, the project setting must refer this directory as an include directory.

#### 4.1.3 Src-tp and Inc-tp directory

The class collection of the third party peripherals. The "third party" means, the outside of the microprocessor.

Currently these directories are not utilized.

#### 4.1.4 murasaki.hpp

Usually, the murasaki.hpp include file is the only one to include from an application program. By including this file, an application can refer all the definition of the Murasaki

This file is stored in the Inc directory.

#### 4.1.5 template directory

#### 4.1.5.1 platform\_config.hpp

The platform\_config.hpp file is a collection of the build configuration. By defining a macro, a programmer can change the behavior of the Murasaki.

There are mainly two types of the configuration in this file.

One type of configuration is to override the murasaki\_config.hpp file. All contents of the murasaki\_config.hpp are macros. These macros are defined to control the Murasaki, for example: the task priority, the task stack size or the timeout period, described in the Definitions and Configuration.

The other configuration type is the assertion inside Murasaki. See MURASAKI\_CONFIG\_NODEBUG for details.

The platform\_config.hpp is better to be copied in the /Inc directory of the application.

#### 4.1.5.2 platform\_defs.hpp

As same as platform\_config.hpp, the platform\_defs.hpp is not the core part of the Murasaki class library. This include file has a definition of the murasaki::platform which provide "nice looking" aggregation of the class objects.

The application programmer can define the murasaki::Platform type freely. There is no limitation or requirement what you put into unless compiler reports an error message.

On the other hand, a programmer may find that adding the peripheral-based class variables and middleware based class variables into the murasaki::Platform type is reasonable. Actually, the independent devices (ie:I2C connected LCD controller) may be better to be a member variable of the mruasaki::Platform type.

The platform\_defs.hpp is better to be copied in the /Inc directory of the application.

See Application Specific Platform as usage sample.

4.2 CubeMX setting 15

### 4.1.5.3 murasaki\_platform.hpp

A header file of the murasaki platform.cpp. This file is better to be copied in the /Inc directory of the application.

### 4.1.5.4 murasaki\_platform.cpp

The murasaki\_platform.cpp is the interface between the application and the HAL/RTOS. This file has variables / functions which user needs to program at porting time.

- · murasaki::platform variable
- · murasaki::debugger variable
- InitPlatform() to initialize the platform variable
- ExecPlatform() to execute the platform algorithm
- · Interrupt routing functions
- HAL assertion function and Custome default exception handler

The murasaki platform.cpp is better to be copied in the /Src directory of the application.

### 4.2 CubeMX setting

There is several required CubeMX setting.

- · Heap Size
- · Stack Size
- Task stack size of the default task

### 4.2.1 Heap Size

Heap is very important in the application with murasaki.

First, class instances are created inside heap region by new operator often. And second, murasaki::Debugger allocates a huge size of FIFO buffer. This buffer stays in between the murasaki::Debugger::Printf() function and the logger task. The size of this FIFO buffer is defined by PLATFORM\_CONFIG\_DEBUG\_BUFFER\_SIZE. The default is 4KB.

Usually, the heap is simply called "heap", without precise definition of terminology. But let's call it "system heap" here. The system heap is the one which is managed by new and delete operators by default.

In addition to the system heap, FreeRTOS has its own heap. This heap is managed separately from the system heap. This management includes the heap size watching and returning error. And this heap is thread safe while the system heap is not.

Using two heap is not easy. And definitely, the FreeRTOS heap is better than the system heap in the embedded application. So, in murasaki, the new and the delete operators are overloaded and redirected to the FreeRTOS heap. See Heap memory consideration for detail.

To avoid the heap allocation problem, it is better to have more than 8kB FreeRTOS heap. The FreeRTOS heap size can be changed by CubeMX:

16 Porting guide

```
Tab => Pinout & Configuration => Middleware => FreeRTOS => Config Parameters Tab => TOTAL_HEAP_SIZE
```

On the other hand, the system heap size can be smaller like 128 Byte because we don't use it..

Note that to know the minimum requirement of the system heap size, you must investigate how much allocations are done before entering FreeRTOS. Because murasaki application doesn't use any system heap, only very small management memory should be required in system heap.

The system Heap size can be set by following place.

```
Tab => Project Manager => Code Generator => Linker Settings
```

### 4.2.2 Stack Size

In this section, the stack means the interrupt stack.

The interrupt stack is used only when the interrupt is accepted. Then, it is basically small.

By the way, murasaki uses its assertion often. Once assertion fails, a message is created by snprintf() function and transmitted through FIFO. These operations consume stack. And assertion can be happen also in the ISR context.

The debugging in the ISR is not easy without assertion and printf(). To make them always possible, it is better to set the interrupt stack size bigger than 256 Bytes. The interrupt stack size can be changed by CubeMX:

```
Tab => Project Manager => Code Generator => Linker Settings
```

### 4.2.3 Task stack size of the default task

The daault task has very small stack (128 Bytes)

This is not enough to use murasaki and its debugger output functionality. It should be increased at smallest 256 Bytes.

It can be changed by CubeMX:

```
Tab => Pinout & Configuration => Middleware => FreeRTOS => Config Parameters Tab => MINIMAL_STACK_SIZE
```

# 4.3 Configuration

Murasaki has configurable parameters.

These parameters control mainly the task size and task priority.

One of the special configurations is MURASAKI\_CONFIG\_NODEBUG macro. This macro controls whether assertion inside Murasaki source code works or ignored.

To customize the configuration, define the configuration macro with the desired value in the platform\_config.hpp file. This definition will override the Murasaki default configuration.

For the detail of each macro, see Definitions and Configuration.

## 4.4 Task Priority and Stack Size

The FreeRTOS task priority is allowed from 1 to configMAX\_PRIORITIES.

Where configMAX\_PRIORITIES is porting dependent. The task with priority == configMAX\_PRIORITIES will run with the highest priority among all tasks.

At the initial state, the Murasaki has two hidden tasks inside. Both are running for the murasaki::Debugger class, and both task's priority are defined as PLATFORM\_CONFIG\_DEBUG\_TASK\_PRIORITY. By default, the value of PL $\leftarrow$  ATFORM\_CONFIG\_DEBUG\_TASK\_PRIORITY is configMAX\_PRIORITIES - 1. That means, debug tasks priority is very high.

The debug tasks should have priority as high as possible. Otherwise, another task may block the debugging message.

Unlike the task priority, the interrupt priority is easy. Usually, it is not so sensitive because the ISR is very short in the good designed RTOS application design. In this case, all ISR can be a same priority.

In the bad designed RTOS application, there are very few things we can do.

## 4.5 Heap memory consideration

In Murasaki, there is a re-definition of operator new and operator delete inside allocators.cpp.

This re-definition let the pvPortMalloc() allocate a fragment of memory for the operator new.

This changes converges all allocation to the FreeRTOS's heap. There is some merit of the convergence:

- The FreeRTOS heap is thread safe while the system heap in SW4STM32 is not thread-safe
- The FreeRTOS heap is checking the heap size limitation and return an error, while the system heap behavior in SW4STM32 is not clear.
- · The heap size calculation is easier if we integrate the memory allocation activity into one heap.

On the other hand, FreeRTOS heap is not able to allocate/deallocate in the ISR context. And it is impossible to use the FreeRTOS heap before starting up the FreeRTOS. Then, we have to follow the rules here:

- C++ new / delete operators have to be called after FreeRTOS started.
- C++ new / delete operators have to be called in the task context.

18 Porting guide

### 4.6 Platform variable

The murasaki::platform and the murasaki::debugger have to be initialized by the InitPlatform() function.

The programming of this function is a responsibility of the porting programmer.

First of all, the porting programmer has to make the peripheral handles as visible from the murasaki platform.cpp.

For example, CubeMx generate the huart2 for Nucleo L152RE for the serial communication over the ST-LINK USB connection. huart2 is defined in main.c as like below:

```
UART_HandleTypeDef huart2;
DMA_HandleTypeDef hdma_usart2_rx;
DMA_HandleTypeDef hdma_usart2_tx;
```

To use this handle, the porting programmer has to declare the same name as an external variable, in the murasaki← \_platform.cpp :

```
extern UART_HandleTypeDef huart2;
```

After these preparations, the porting programmer can program the InitPlatform():

```
void InitPlatform()
    // UART device setting for console interface.
    // On Nucleo, the port connected to the USB port of ST-Link is
    // referred here.
    murasaki::platform.uart_console = new
      murasaki::Uart(&huart2);
    // UART is used for logging port
    // At least one logger is needed to run the debugger class.
    murasaki::platform.logger = new murasaki::UartLogger(
     murasaki::platform.uart_console);
    // Setting the debugger
    murasaki::debugger = new murasaki::Debugger(
      nurasaki::platform.logger);
    \ensuremath{//} Set the debugger as AutoRePrint mode, for the easy operation.
    murasaki::debugger->AutoRePrint(); // type any key to show history.
    // For demonstration, one GPIO LED port is reserved.
    // The port and pin names are fined by CubeMX.
    murasaki::platform.led = new murasaki::BitOut(LD2_GPIO_Port,
     LD2_Pin);
}
```

In this sample, we initialize the uart\_console member variable which is AbstractUart class. The application programmer control the UART2 over this uart\_console member variable.

In the second step, we pass this uart\_cosole to the logger member variable. This member variable is an essential stub for the murasaki::debugger. In this example, we assign the UART2 port as interface for the debugging output.

After the logger becomes ready, we initialize the murasaki::debugger. As we already discussed, this debugger receives a logger object as a parameter. The debugger output all messages through this logger.

The last step is optional. We invoke the murasaki::Debugger::AutoRePrint() member function. By calling this function, logger re-print the old data in the FIFO again whenever the end-user type any key of the keyboard.

This "auto re-print by any key" is convenient in the small system. But for the large system which has its own command line shell, this input-interruption is harmful. For such the system, programmer want to call murasaki::

Debugger::RePrint() member function, by certain customer command.

Once the debugger is ready to use, we create the led member variable as a general purpose output port of the application .

The ExecPlatform() function implements the actual algorithm of application. In the example below, the application is blinking a LED and printing a messages on the console output.

4.7 Routing interrupts 19

```
void ExecPlatform()
{
    // counter for the demonstration.
    static int count = 0;

    // Loop forever
    while (true) {
            // Toggle LED.
            murasaki::platform.led->Toggle();

            // print a message with counter value to the console.
            murasaki::debugger->Printf("Hello %d \n\r", count);

            // update the counter value.
            count++;

            // wait for a while
            murasaki::Sleep(static_cast<murasaki::WaitMilliSeconds>(500));
    }
}
```

Finally, above two functions have to be called from StartDefaultTask of the main.c. Also, main.c must include the murasaki\_platform.hpp to read the prototype of these functions.

Following is the sample of the StartDefaultTask(). The actual code have a comment to work together the code generator of the CubeMX. But this sample remove them because of the documenattion tool (doxygen) limitation.

```
void StartDefaultTask(void const * argument)
{
    InitPlatform();
    ExecPlatform();

    for(;;)
    {
        osDelay(1);
    }
}
```

### 4.7 Routing interrupts

The murasaki\_platform.cpp has skeletons of HAL callback.

These callbacks are pre-defined inside HAL as receptors of interrupt. These definitions inside HAL are "weak" binding. Thus, these skeletons in murasaki\_platform.cpp overrides the definition. The porting programmer have to program these skeltons correctly.

In the Murasaki manner, the skeletons have to call the relevant callback member function of platform variables. For example, this is the typical programming of the call back :

```
void HAL_UART_TxCpltCallback(UART_HandleTypeDef * huart)
{
    if (murasaki::platform.uart_console->TransmitCompleteCallback(huart))
        return;
}
```

In this sample, the TxCpltCallback() calles murasaki::platform.uart\_console->TransmitCompleteCallback() member function. And then return if that member function returns true. Note that all the callacks in the Murasaki class returns true if the given peripheral handle matches with its internal handle. Thus, this is good way to poll all the UART peripheral inside this callback function.

Following is the list of the interrupts which application have to route to the peripehral class variables.

20 Porting guide

```
void HAL_UART_TxCpltCallback(UART_HandleTypeDef * huart);
void HAL_UART_RxCpltCallback(UART_HandleTypeDef * huart);
void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart);
void HAL_SPI_TxRxCpltCallback(SPI_HandleTypeDef *hspi);
void HAL_SPI_ErrorCallback(SPI_HandleTypeDef * hspi);
void HAL_I2C_MasterTxCpltCallback(I2C_HandleTypeDef * hi2c);
void HAL_I2C_MasterRxCpltCallback(I2C_HandleTypeDef * hi2c);
void HAL_I2C_SlaveTxCpltCallback(I2C_HandleTypeDef * hi2c);
void HAL_I2C_SlaveRxCpltCallback(I2C_HandleTypeDef * hi2c);
void HAL_I2C_ErrorCallback(I2C_HandleTypeDef * hi2c);
void HAL_I2C_ErrorCallback(I2C_HandleTypeDef * hi2c);
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_P);
```

## 4.8 Error handling

The murasaki platform.cpp has two error handling functions.

These functions are pre-programmed from the first. And usually its enough to use the pre-programmed version. In the other hand the porting programmer have to modify the application program to call these error handling functions at appropriate situation. Otherwise, these error handling functions will be never called.

The CustomAssertFailed() function should be called from the assert\_failed() function. The assert\_failed() function is located in the main.c. Modifying the assert\_failed() is the responsibility of the porting programmer.

```
void assert_failed(uint8_t* file, uint32_t line)
{
    CustomAssertFailed(file, line);
}
```

To enable the assert\_failed(), the porting programmer have to uncomment the USE\_FULL\_ASSERT macro inside stm32xxxx\_hal\_conf.h. The file name is depend on the target microprocessor. Thus, the porting programmer have to search the all files inside project.

At the time of 2019/May, this definition is in the one for the following files :

- stm32f0xx\_hal\_conf.h
- stm32f3xx\_hal\_conf.h
- stm32f7xx\_hal\_conf.h
- stm32l1xx\_hal\_conf.h

The CustomDefaultHandler() function should be called from the default exception routine. But the system default exception handler ( Default\_Handler ) doesn't do anything by default. To maximize the information to the JTAG debugger, this is programmed as very simple eternal loop.

The default exception handler can be programmed or left untouched as porting programmer want. It is up to the system policy. If it is re-programmed to call the CustomDefaultHandler(), murasaki::debugger object take the control of the debug message FIFO at the exception handler context.

If the exception happened and the CustomDefaultHandler is called, the end user can see the entire messages in the debug FIFO by typing any key of the keyboard. This is useful to see the last message from the assertion. The last message usually represent the cause of the exception. The end user can debug the application program based on this last assertion message.

The HAL default exception routine is programmed at startup/startup\_stm32xxxxx.s by assembly language.

The porting programmer can modify it as below, to call the CustomDefaultHandler();

```
Default_Handler:
Infinite_Loop:
bl CustomDefaultHandler
b Infinite_Loop
.size Default_Handler, .-Default_Handler
```

## 4.9 Summary of the porting

Following is the porting steps:

- · Adjust heap size and stack size as described in the CubeMX setting
- · Generate an application skeleton from CubeMX.
- · Checkout Murasaki repository into your project.
- · Copy the template files as described in the Directory Structure .
- · Configure Muraaski as described in the Configuration and the Task Priority and Stack Size
- Call InitPlatform() and ExecPlatform() as described Platform variable.
- Route the interrupts as described Routing interrupts.
- Route the error handling as described Error handling

22 Porting guide

# **Chapter 5**

# **Module Index**

# 5.1 Modules

## Here is a list of all modules:

Murasaki Class Collection
Synchronization and Exclusive access
Third party classes
Definitions and Configuration
Application Specific Platform
Abstract Classes
Helper classes
CMSIS
Stm32f7xx_system
STM32F7xx_System_Private_Includes
STM32F7xx_System_Private_TypesDefinitions
STM32F7xx_System_Private_Defines
STM32F7xx_System_Private_Macros
STM32F7xx_System_Private_Variables
STM32F7xx_System_Private_FunctionPrototypes
STM32F7xx System Private Functions

24 Module Index

# **Chapter 6**

# Namespace Index

6.1	Namespa	ce List

murasaki													
Personal Platform parts collection		 		 	 						 		67

26 Namespace Index

# **Chapter 7**

# **Hierarchical Index**

# 7.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

murasaki::AudioCodecStrategy	75
murasaki::Adau1361	71
murasaki::CriticalSection	85
murasaki::Debugger	
murasaki::FifoStrategy	
murasaki::DebuggerFifo	
murasaki::GPIO_type	
murasaki::LoggerStrategy	116
murasaki::UartLogger	147
murasaki::LoggingHelpers	
murasaki::PeripheralStrategy	
murasaki::BitInStrategy	
murasaki::Bitln	
murasaki::BitOutStrategy	
murasaki::BitOut	
murasaki::I2CMasterStrategy	
murasaki::I2cMaster	
murasaki::l2cSlaveStrategy	
murasaki::I2cSlave	
murasaki::SpiMasterStrategy	
murasaki::SpiMaster	
murasaki::SpiSlaveStrategy	
murasaki::SpiSlave	
murasaki::UartStrategy	
murasaki::DebuggerUart	
murasaki::Uart	
murasaki::Platform	
murasaki::SpiSlaveSpecifierStrategy	
murasaki::SpiSlaveSpecifier	
murasaki::Synchronizer	
murasaki::TaskStrategy	
murasaki::Task	137

28 Hierarchical Index

# **Chapter 8**

# **Class Index**

## 8.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

murasaki::Adau1361	
Audio Codec LSI class	71
murasaki::AudioCodecStrategy	
Abstract audio codec controller	75
murasaki::BitIn	
General purpose bit input	78
murasaki::BitInStrategy	
Definition of the root class of bit input	80
murasaki::BitOut	
General purpose bit output	81
murasaki::BitOutStrategy	
Definition of the root class of bit output	83
murasaki::CriticalSection	
A critical section for task context	85
murasaki::Debugger	
Debug class. Provides printf() style output for both task and ISR context	86
murasaki::DebuggerFifo	
FIFO with thread safe	89
murasaki::DebuggerUart	
Logging dedicated UART class	91
murasaki::FifoStrategy	
Basic FIFO without thread safe	96
murasaki::GPIO_type	
A structure to en-group the GPIO port and GPIO pin	97
murasaki::l2cMaster	
Thread safe, blocking IO. Encapsulating I2C master. Based on STM32Cube HAL driver and	
FreeRTOS	98
murasaki::I2CMasterStrategy	
Definition of the root class of I2C master	104
murasaki::l2cSlave	
Thread safe, blocking IO. Encapsulating I2C slave. Based on STM32Cube HAL driver and Free ←	
RTOS	108
murasaki::l2cSlaveStrategy	
Definition of the root class of I2C Slave	113
murasaki::LoggerStrategy	
Abstract class for logging	116

30 Class Index

murasaki::LoggingHelpers	
A stracture to engroup the logging tools	118
murasaki::PeripheralStrategy	
Mother of all peripheral class	118
murasaki::Platform	
Custom aggregation struct for user platform	119
murasaki::SpiMaster	
Thread safe, blocking IO. Encapsulating SPI master. Based on STM32Cube HAL driver and	
FreeRTOS	120
murasaki::SpiMasterStrategy	
Root class of the SPI master	123
murasaki::SpiSlave	
Thread safe, blocking IO. Encapsulating SPI slave. Based on STM32Cube HAL driver and Free ←	
RTOS	126
murasaki::SpiSlaveSpecifier	
A speficier of SPI slave	129
murasaki::SpiSlaveSpecifierStrategy	
Definition of the root class of SPI slave specifier	132
murasaki::SpiSlaveStrategy	
Root class of the SPI slave	134
murasaki::Synchronizer	
Synchronization class between a task and interrupt. This class provide the synchronization	
between a task and interrupt	136
murasaki::Task	
An easy to use task class	137
murasaki::TaskStrategy	
A mother of all tasks	139
murasaki::Uart	
Concrete implementation of UART controller. Based on the STM32Cube HAL DMA Transfer	141
murasaki::UartLogger	
Logging through an UART port	147
murasaki::UartStrategy	
Definition of the root class of LIART	149

# **Chapter 9**

# File Index

# 9.1 File List

Here is a list of all documented files with brief descriptions:

/home/takemasa/murasaki_samples/nucleo-f746-sample/Inc/main.h	
: Header for main.c file. This file contains the common defines of the application	155
/home/takemasa/murasaki_samples/nucleo-f746-sample/Inc/murasaki_platform.hpp	
An interface for the application from murasaki library to main.c	157
/home/takemasa/murasaki_samples/nucleo-f746-sample/Inc/platform_config.hpp	
Application dependent configuration	158
/home/takemasa/murasaki_samples/nucleo-f746-sample/Inc/platform_defs.hpp	
Murasaki platform customize file	159
/home/takemasa/murasaki_samples/nucleo-f746-sample/lnc/stm32f7xx_hal_conf.h	
HAL configuration file	160
/home/takemasa/murasaki_samples/nucleo-f746-sample/lnc/stm32f7xx_it.h	
This file contains the headers of the interrupt handlers	166
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc-tp/adau1361.hpp	167
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/audiocodecstrategy.hpp	168
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/bitin.hpp	
GPIO bit in class	169
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/bitinstrategy.hpp	
Abstract class of the GPIO bit in	171
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/bitout.hpp	
GPIO bit out class	173
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/bitoutstrategy.hpp	
Abstract class of GPIO bit out	175
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/criticalsection.hpp	
Class to protect a certain section from the interference	177
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/debugger.hpp	
Debug print class. For both ISR and task	178
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/debuggerfifo.hpp	
Dedicated FIFO to logging the debug message	179
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/debuggeruart.hpp	
UART. Thread safe and blocking IO	181
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/fifostrategy.hpp	
Abstract class of FIFO	183
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/i2cmaster.hpp	
I2C master. Thread safe, blocking IO	184
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/i2cmasterstrategy.hpp	
Root class definition of the I2C Master	186

32 File Index

/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/i2cslave.hpp	
I2C slave. Thread safe, blocking IO	188
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/i2cslavestrategy.hpp  Root class definition of the I2C Slave	190
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/loggerstrategy.hpp	
Simplified logging function	192
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/murasaki.hpp Application include file for Murasaki class library	194
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/murasaki_0_intro.hpp  Doxygen document file. No need to include	195
/home/takemasa/murasaki samples/nucleo-f746-sample/murasaki/lnc/murasaki 1 env.hpp	
Doxygen document file. No need to include	195
Doxygen document file. No need to include	195
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/murasaki_3_pg.hpp Porting Guide	196
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/murasaki_4_mod.hpp	190
Module definition	196
Assertion definition	196
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/murasaki_config.hpp  Configuration file for platform	198
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/murasaki_defs.hpp	190
Common definition of the platfrom	199
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/murasaki_syslog.hpp Syslog definition	200
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/peripheralstrategy.hpp  Mother of All peripheral	202
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/spimaster.hpp	
SPI Master. Thread safe and blocking IO	203
SPI master root class	205
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/spislave.hpp SPI Slave. Thread safe and blocking IO	207
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/spislavespecifier.hpp STM32 SPI slave speifire	209
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/spislavespecifierstrategy.hpp	200
Abstract class of SPI slave specification	211
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/spislavestrategy.hpp  SPI master root class	213
/home/takemasa/murasaki samples/nucleo-f746-sample/murasaki/lnc/synchronizer.hpp	
Synchronization between a Task and interrupt	215
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/task.hpp Simplified Task class	216
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/taskstrategy.hpp	
Mother of All Tasks	218
UART. Thread safe and blocking IO	219
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Inc/uartlogger.hpp  Logging to Uart	221
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/uartstrategy.hpp  Root class definition of the UART driver	222
/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/Src/allocators.cpp	
Alternative memory allocators	224
/home/takemasa/murasaki_samples/nucleo-f746-sample/Src/main.c : Main program body	225
/home/takemasa/murasaki_samples/nucleo-f746-sample/Src/murasaki_platform.cpp	
A glue file between the user application and HAL/RTOS	229

9.1 File List

/home/takemasa/murasaki_samples/nucleo-f746-sample/Src/stm32f7xx_it.c	
Interrupt Service Routines	231
/home/takemasa/murasaki_samples/nucleo-f746-sample/Src/system_stm32f7xx.c	
CMSIS Cortex-M7 Device Peripheral Access Layer System Source File	232

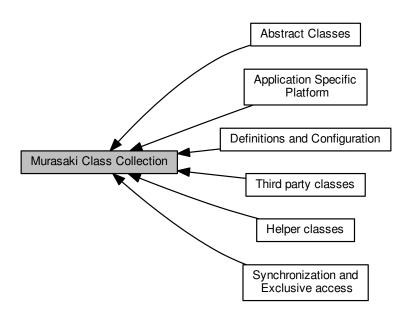
34 File Index

# Chapter 10

# **Module Documentation**

## 10.1 Murasaki Class Collection

Collaboration diagram for Murasaki Class Collection:



### **Modules**

- Synchronization and Exclusive access
- · Third party classes
- · Definitions and Configuration
- Application Specific Platform
- Abstract Classes
- · Helper classes

### Classes

- class murasaki::BitIn
- · struct murasaki::GPIO\_type
- · class murasaki::BitOut
- · class murasaki::Debugger
- · class murasaki::I2cMaster
- class murasaki::I2cSlave
- · class murasaki::SpiMaster
- · class murasaki::SpiSlave
- · class murasaki::SpiSlaveSpecifier
- · class murasaki::Task
- · class murasaki::Uart
- · class murasaki::UartLogger

### **Macros**

- #define MURASAKI\_ASSERT(COND)
- #define MURASAKI PRINT ERROR(ERR)
- #define MURASAKI\_SYSLOG(FACILITY, SEVERITY, FORMAT, ...)

### 10.1.1 Detailed Description

This is a reference guide of murasaki class library. This guide describes class by class and cover entire library. It is not recommended to read the reference for the first time user.

Alternatively, the Usage Introduction is provided to study step by step.

### 10.1.2 Macro Definition Documentation

### 10.1.2.1 #define MURASAKI\_ASSERT( COND )

### Value:

Assert the COND is true.

### **Parameters**

COND	Condition as bool type.
------	-------------------------

Print the COND expression to the logging port if COND is false. Do nothing if CODN is true.

After printing the assertion failure message, currently running task is suspended. If it is the interrupt context, just continue the processing.

This assertion do nothing if programmer defines MURASAKI\_CONFIG\_NODEBUG macro as true. This macro is defined in the file platform\_config.hpp.

### 10.1.2.2 #define MURASAKI\_PRINT\_ERROR( ERR )

### Value:

Print ERR if ERR is true.

#### **Parameters**

ERR	Condition as bool type.
-----	-------------------------

Print the ERR expression to the logging port if COND is true. Do nothing if ERR is true.

This assertion do nothing if programmer defines MURASAKI\_CONFIG\_NODEBUG macro as true. This macro is defined in the file platform\_config.hpp.

For example, following code is typical usage of this macro. ERROR maccro is copied from STM32Cube HAL source code.

```
1 bool Uart::HandleError(void* const ptr)
2
        MURASAKI_ASSERT (nullptr != ptr)
4
5
        if (peripheral_ == ptr) {
             // Check error, and print if exist.
MURASAKI_PRINT_ERROR(peripheral_->ErrorCode & HAL_UART_ERROR_DMA);
MURASAKI_PRINT_ERROR(peripheral_->ErrorCode & HAL_UART_ERROR_PE);
6
             MURASAKI_PRINT_ERROR(peripheral_->ErrorCode & HAL_UART_ERROR_NE);
10
              MURASAKI_PRINT_ERROR(peripheral_->ErrorCode & HAL_UART_ERROR_FE);
              MURASAKI_PRINT_ERROR(peripheral_->ErrorCode & HAL_UART_ERROR_ORE);
MURASAKI_PRINT_ERROR(peripheral_->ErrorCode & HAL_UART_ERROR_DMA);
11
12
13
              return true;
                                   // report the ptr matched
14
15
16
              return false; // report the ptr doesn't match
17
18 }
```

### 10.1.2.3 #define MURASAKI\_SYSLOG( FACILITY, SEVERITY, FORMAT, ... )

output The debug message

### **Parameters**

FACILITY	Specify which facility makes this log. Choose from murasaki::SyslogFacility	
SEVERITY	Specify how message is severe. Choose from murasaki::SyslogSeverity	
FORMAT	Message format as printf style.	

Output the debugg message to debug console output.

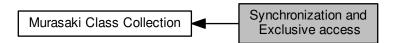
The output message is filtered by the internal thereshold set by murasaki::SetSyslogSererityThreshold, murasaki::SetSyslogFacilityMask and murasaki::AddSyslogFacilityToMask. See these function's document to understand how filter works.

There is recommendation in the SEVERITY parameter :

- murasaki::kseDebug for Development/Debug message for tracing normal operation.
- murasaki::kseWarning for relatively severe condition which need abnormal action, or cannot handle.
- murasaki::kseError for falty condtion from HAL or hardware.
- murasaki::kseEmergency for software logic error like assert fail

# 10.2 Synchronization and Exclusive access

Collaboration diagram for Synchronization and Exclusive access:



### Classes

- · class murasaki::CriticalSection
- class murasaki::Synchronizer

## 10.2.1 Detailed Description

These classes are used as parts of the other classes.

# 10.3 Third party classes

Collaboration diagram for Third party classes:



## Classes

• class murasaki::Adau1361

## 10.3.1 Detailed Description

## 10.4 Definitions and Configuration

Collaboration diagram for Definitions and Configuration:



- #define PLATFORM\_CONFIG\_DEBUG\_LINE\_SIZE 256
- #define PLATFORM\_CONFIG\_DEBUG\_BUFFER\_SIZE 4096
- #define PLATFORM CONFIG DEBUG SERIAL TIMEOUT (murasaki::kwmsIndefinitely)
- #define PLATFORM\_CONFIG\_DEBUG\_TASK\_STACK\_SIZE 256
- #define PLATFORM\_CONFIG\_DEBUG\_TASK\_PRIORITY (( configMAX\_PRIORITIES-1 > 0 ) ? configM
   AX PRIORITIES-1 : 0)
- #define MURASAKI\_CONFIG\_NODEBUG false

### 10.4.1 Detailed Description

### 10.4.2 Macro Definition Documentation

### 10.4.2.1 #define MURASAKI\_CONFIG\_NODEBUG false

Surpress MURASAKI ASSERT macro.

Set this macro to true, to discard the assertion MURASAKI\_ASSERT. Set this macro false, to use the assertion.

To override the definition here, define same macro inside platform\_config.hpp.

### 10.4.2.2 #define PLATFORM\_CONFIG\_DEBUG\_BUFFER\_SIZE 4096

Size[byte] of the circular buffer to be transmitted through the serial port.

The circular buffer array length to copy the formatted strings before transmitting through the uart.

To override the definition here, define same macro inside platform config.hpp.

### 10.4.2.3 #define PLATFORM\_CONFIG\_DEBUG\_LINE\_SIZE 256

Size of one line[byte] in the debug printf.

The array length to store the formatted string. Note that this array is a private instance variable. Then, it will occupy the memory where the class is instantiated. For example, if an object is instantiated in the heap, this line buffer will be reserved in the heap.

If the class is instantiated on the stack, the buffer will be reserved in the stack.

To override the definition here, define same macro inside platform\_config.hpp.

10.4.2.4 #define PLATFORM\_CONFIG\_DEBUG\_SERIAL\_TIMEOUT (murasaki::kwmsIndefinitely)

Timeout of the serial port to transmit the string through the Debug class.

By default, there is no timeout. Wait for eternally.

To override the definition here, define same macro inside platform config.hpp.

10.4.2.5 #define PLATFORM\_CONFIG\_DEBUG\_TASK\_PRIORITY (( configMAX\_PRIORITIES-1 > 0 ) ? configMAX\_PRIORITIES-1 : 0)

The task proiority of the debug task.

The priority of the murasaki::Debuger internal task. To output the logging data as fast as possible, the debug taks have to have relatively high priority. In other hand, to yield the CPU to the critical tasks, it's priority have to be smaller than the max priority.

To override the definition here, define same macro inside platform\_config.hpp.

10.4.2.6 #define PLATFORM\_CONFIG\_DEBUG\_TASK\_STACK\_SIZE 256

Size[Byte] of the task inside Debug class.

The murasaki::Debuger class has internal task to handle its FIFO buffer.

To override the definition here, define same macro inside platform config.hpp.

### 10.4.3 Enumeration Type Documentation

10.4.3.1 enum murasaki::I2cStatus

Return status of the I2C classes.

This enums represents the return status from the I2C class method.

In a single master controler system, you need to care only ki2csNak and ki2csTimeOut. Other error may be caused by multiple master system.

The ki2csNak is returned when one of two happens:

- · The slave device terminated transfer.
- · No slave device responded to the address specified by master device.

The ki2csTimeOUt is returned when slave device streched transfere too long.

The ki2csArbitrationLost is returned when another master won the arbitration. Usulally, the master have to re-try the transfer after certain waiting period.

The ki2csBussError is fatal condition. In the master mode, it could be problem of other device. The root cause is not deterministic. Probably it is hardware problem.

### Enumerator

ki2csOK ki2csOK

ki2csTimeOut Master mode error. No response from device.

ki2csNak Master mode error. Device answeres NAK.

ki2csBussError Master&Slave mode error. START/STOP condition at irregular location.

ki2csArbitrationLost Master&Slave mode error. Lost arbitration against other master device.

ki2csOverrun Slave mode error. Overrun or Underrun was detected.

ki2csDMA Some error detected in DMA module.

ki2csUnknown Unknown error.

### 10.4.3.2 enum murasaki::SpiClockPhase

SPI clock configuration for master.

This enum represents the setting of the SPI PHA bit of the master configuration. The PHA setting 0 and 1 is LatchThenShift and ShiftThenLatch respectively.

#### Enumerator

ksphLatchThenShiftkscpLatchThenShift PHA=0. The first edge is latching. The second edge is shifting.ksphShiftThenLatchkscpShiftThenLatch PHA = 1. The first edge is shifting. The second edge is latching.

### 10.4.3.3 enum murasaki::SpiClockPolarity

SPI clock configuration for Master.

This enum represents the setting of the SPI POL bit of the master configuration. The POL setting 0/1 is RiseThenFall and Fall thenRise respectively.

### Enumerator

```
kspoRiseThenFall kscpRiseThenFall POL = 0
kspoFallThenRise kscpFallThenrise POL = 1
```

### 10.4.3.4 enum murasaki::SpiStatus

Return status of the SPI classes.

This enums represents the return status of from the SPI class method.

kspisModeFault is returned when the NSS pins are aserted. Note that the Murasaki library doesn't support the Multi master SPI operation. So, this is fatal condition.

kpisOverflow and the kpisDMA are fatal condition. These can be the problem of the lower driver problem.

### **Enumerator**

kspisOK ki2csOK

kspisTimeOut Master mode error. No response from device.

kspisModeFault SPI mode fault error. Two master corrision.

kspisModeCRC CRC protocol error.

kspisOverflow Over run.

kspisFrameError Error on TI frame mode.

kspisDMA DMA error.

kspisErrorFlag Other error flag.

kspisAbort Problem in abort process. No way to recover.

kspisUnknown Unknown error.

### 10.4.3.5 enum murasaki::SyslogFacility

Category to filter the Syslog output.

These are independent facilities to filter the Syslog message output. Each module should specify appropriate facility.

Internally, these value will be used as bit position in mask.

### Enumerator

kfaKernel is specified when the message is bound with the kernel issue.

kfaSerial is specified when the message is from the serial module.

kfaSpiMaster kfaSpi is specified when the message is from the SPI master module

kfaSpiSlave kfaSpi is specified when the message is from the SPI slave module

*kfal2cMaster* kfal2c is specified when the message is from the I2C master module.

kfal2cSlave kfal2c is specified when the message is from the I2C slave module.

kfal2s kfal2s is specified when the message is from the I2S module

kfaSai is specified when the message is from the SAI module.

kfaLog kfaLog is specified when the message is from the logger and debugger module.

kfaNone Disable all facility.

kfaAll Enable all facility.

kfaUser0 User defined facility.

kfaUser1 User defined facility.

kfaUser2 User defined facility.

kfaUser3 User defined facility.

kfaUser4 User defined facility.

kfaUser5 User defined facility.

kfaUser6 User defined facility.

kfaUser7 User defined facility.

### 10.4.3.6 enum murasaki::SyslogSeverity

Message severity level.

The lower value is the more serious condition.

### **Enumerator**

**kseEmergency** kseEmergency means the system is unusable.

kseAlert means some acution must be taken immediately.

kseCritical kseCritical means critical condition.

kseError means error conditions.

kseWarning kseWarning means warning condition.

**kseNotice** kseNotice means normal but significant condition.

kselnfomational kselnfomational means infomational message.

kseDebug kseDebug means debug-level message

### 10.4.3.7 enum murasaki::UartHardwareFlowControl

Attribute of the UART Hardware Flow Control.

This is dedicated to the UartStrategy class.

### **Enumerator**

kuhfcNone No hardware flow control.

kuhfcCts Control CTS, but RTS.kuhfcRts Control RTS, but CTS.

kuhfcCtsRts Control Both CTS and RTS.

### 10.4.3.8 enum murasaki::UartStatus

Return status of the UART classes.

The Parity error and the Frame error may occur when user connects DCT/DTE by different communication setting.

The Noise error may cuase by the noise on the line.

The overrun may cause when the DMA is too slow or hand shake is not working well.

The DMA error may cause some problem inisde HAL.

### Enumerator

kursOK No error.

kursTimeOut Time out during transmission / receive.

kursParity Parity error.

kursNoise Error by Noise.

kursFrame Frame error.

kursOverrun Overrun error.

kursDMA Error inside DMA module.

### 10.4.3.9 enum murasaki::UartTimeout

This is specific enum for the AbstractUart::Receive() to specify the use of idle line timeout.

The idle line time out is dedicated function of the STM32 peripherals. The interrrupt happens when the receive data is discontinued certain time.

### **Enumerator**

kutNoldleTimeout kutNoldleTimeout is specified when API should has normal timeout.

kutldleTimeout kutldleTimeout is specified when API should time out by Idle line

10.4.3.10 enum murasaki::WaitMilliSeconds: uint32\_t

Wait time by milliseconds. For the function which has "wait" or "timeout" parameter.

An uint32\_t derived type for specifying wait duration. The integer value represents the waiting duration by miliseconds. Usually a value of this type is passed to some functions as parameter. There are two special cases.

kwmsPolling means function will return immediately regardless of waited event.In other word, with this parameter, function causes time out immediately. Some function may provides the way to know what was the status of the waited event. But some may not.

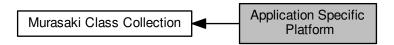
kwmsIndefinitely means function will will not cause time out.

### Enumerator

kwmsPolling Not waiting. Immediate timeout.kwmsIndefinitely Wait forever.

## 10.5 Application Specific Platform

Collaboration diagram for Application Specific Platform:



### **Classes**

· struct murasaki::Platform

### **Functions**

- · void InitPlatform ()
- void ExecPlatform ()
- void CustomAssertFailed (uint8 t \*file, uint32 t line)
- void CustomDefaultHandler ()
- void HAL\_UART\_TxCpltCallback (UART\_HandleTypeDef \*huart)
- void HAL\_UART\_RxCpltCallback (UART\_HandleTypeDef \*huart)
- void HAL\_UART\_ErrorCallback (UART\_HandleTypeDef \*huart)
- void HAL\_SPI\_TxRxCpltCallback (SPI\_HandleTypeDef \*hspi)
- void HAL\_SPI\_ErrorCallback (SPI\_HandleTypeDef \*hspi)
- void HAL I2C MasterTxCpltCallback (I2C HandleTypeDef \*hi2c)
- void HAL\_I2C\_SlaveTxCpltCallback (I2C\_HandleTypeDef \*hi2c)
- void HAL\_I2C\_ErrorCallback (I2C\_HandleTypeDef \*hi2c)
- void HAL\_GPIO\_EXTI\_Callback (uint16\_t GPIO\_P)

### **Variables**

Debugger \* murasaki::debugger

### 10.5.1 Detailed Description

Typical usage of these variables can be seen below. First of all, an .cpp file have to include murasaki.hpp.

```
#include "murasaki.hpp"
```

And then, define the murasaki::debugger in the global context. Note that this is essential to use certain debug macros.

The definition of the murasaki::platform is optional. But it is recommended to declare for the ease of reading.

```
murasaki::Debugger * murasaki::debugger;
murasaki::Platform * murasaki::platform;
```

Finally, initialize the murasaki::debugger and murasaki::platform. Again, the murasaki::debugger is essential to use the debug macro. The debug macros are used inside murasaki class library. Then, it is mandatory to initialize the debugger member variable.

The following code fragment initialize only the debugger related member variables. Also, the murasaki::Platform variable is refereed.

The platfrom.uart\_console member variable hooks a murasaki::AbstractUart class variable. In this sample, The murasaki::Uart class is instantiated. The Uart constructor receives the pointer to the UART\_HandleTypeDef. Usually, the UART\_HandleTypeDef variable is generated by CubeMX. For example, "huart3" variable in the main.c file.

The platform.logger member variable hooks a murasaki::AbstractLogger variable. In this example, murasaki::Uart ← Logger class variable is instantiated.

Finally, the debugger variable is initialized. The murasaki::Debugger constructor receives murasaki::AbstractLogger \* type.

```
void InitPlatform(UART_HandleTypeDef * uart_handle)
{
   murasaki::platform.uart_console = new murasaki::Uart(uart_handle);
   murasaki::platform.logger = new murasaki::UartLogger(murasaki::platform.uart_console);

   murasak::debugger = new murasaki::Debugger(murasaki::platform.logger);
}
```

### 10.5.2 Function Documentation

```
10.5.2.1 void CustomAssertFailed ( uint8_t * file, uint32_t line )
```

Hook for the assert\_failure() in main.c.

### **Parameters**

file	Name of the source file where assertion happen
line	Number of the line where assertion happen

This routine provides a custom hook for the assertion inside STM32Cube HAL. All assertion raised in HAL will be redirected here.

```
1 void assert_failed(uint8_t* file, uint32_t line)
2 {
3          CustomAssertFailed(file, line);
4 }
```

By default, this routine output a message with location informaiton to the debugger console.

```
10.5.2.2 void CustomDefaultHandler ( )
```

Hook for the default exception handler. Never return.

This routine is invoked from the default handler of the start up file. The modification to the startup file is user's responsibility.

For example, the start up code for the Nucleo-L152RE is startup\_stml152xe.s. This file is generated by CubeMX. This file has default handler as like this:

```
1 .section .text.Default_Handler,"ax",%progbits
2    Default_Handler:
3    Infinite_Loop:
4    b Infinite_Loop
```

This code can be modified to call CustomDefaultHanler as like this:

```
1 .global CustomDefaultHandler
2 .section .text.Default_Handler,"ax",%progbits
3 Default_Handler:
4    bl CustomDefaultHandler
5 Infinite_Loop:
6    b Infinite_Loop
```

10.5.2.3 void ExecPlatform ( )

The body of the real application.

The body function of the murasaki application. Usually this function is called from the StartDefaultTask() of the main.c.

This function is invoked only once, and never return. See InitPlatform() as calling sample.

By default, it toggles LED as sample program. Inside this function can be customized freely.

```
10.5.2.4 void HAL_GPIO_EXTI_Callback ( uint16_t GPIO_P )
```

Optional interrupt handling of EXTI.

### **Parameters**

GPIO⇔	Pin number from 0 to 31
_ <i>P</i>	

This is called from inside of HAL when an EXTI is accepted.

STM32Cube HAL has same name function internally. That function is invoked whenever an relevant interrupt happens. In the other hand, that function is declared as weak bound. As a result, this function overrides the default error interrupt call back.

The GPIO\_P is the number of Pin. If programmmer set the pin name by CubeMX as FOO, the macro to identify that EXTI is FOO\_PIN

10.5.2.5 void HAL\_I2C\_ErrorCallback ( I2C\_HandleTypeDef \* hi2c )

Optional error handling of I2C.

#### **Parameters**

hi2c

This is called from inside of HAL when an I2C error interrupt is accepted.

STM32Cube HAL has same name function internally. That function is invoked whenever an relevant interrupt happens. In the other hand, that function is declared as weak bound. As a result, this function overrides the default error interrupt call back.

In this call back, the uart device handle have to be passed to the murasaki::I2c::HandleError() function.

10.5.2.6 void HAL\_I2C\_MasterTxCpltCallback ( I2C\_HandleTypeDef \* hi2c )

Essential to sync up with I2C.

### **Parameters**

hi2c

This is called from inside of HAL when an I2C transmission done interrupt is accepted.

STM32Cube HAL has same name function internally. That function is invoked whenever an relevant interrupt happens. In the other hand, that function is declared as weak bound. As a result, this function overrides the default TX interrupt call back.

In this call back, the uart device handle have to be passed to the murasaki::I2c::TransmitCompleteCallback() function.

10.5.2.7 void HAL\_I2C\_SlaveTxCpltCallback ( I2C\_HandleTypeDef \* hi2c )

Essential to sync up with I2C.

### **Parameters**

hi2c

This is called from inside of HAL when an I2C transmission done interrupt is accepted.

STM32Cube HAL has same name function internally. That function is invoked whenever an relevant interrupt happens. In the other hand, that function is declared as weak bound. As a result, this function overrides the default TX interrupt call back.

In this call back, the I2C slave device handle have to be passed to the murasaki::I2cSlave::TransmitComplete ← Callback() function.

10.5.2.8 void HAL\_SPI\_ErrorCallback ( SPI\_HandleTypeDef \* hspi )

Optional error handling of SPI.

#### **Parameters**

hspi

This is called from inside of HAL when an SPI error interrupt is accepted.

STM32Cube HAL has same name function internally. That function is invoked whenever an relevant interrupt happens. In the other hand, that function is declared as weak bound. As a result, this function overrides the default error interrupt call back.

In this call back, the uart device handle have to be passed to the murasaki::Uart::HandleError() function.

10.5.2.9 void HAL\_SPI\_TxRxCpltCallback ( SPI\_HandleTypeDef \* hspi )

Essential to sync up with SPI.

#### **Parameters**



This is called from inside of HAL when an SPI transfer done interrupt is accepted.

STM32Cube HAL has same name function internally. That function is invoked whenever an relevant interrupt happens. In the other hand, that function is declared as weak bound. As a result, this function overrides the default TX/RX interrupt call back.

In this call back, the SPI device handle have to be passed to the murasaki::Spi::TransmitAndReceiveComplete ← Callback () function.

10.5.2.10 void HAL\_UART\_ErrorCallback ( UART\_HandleTypeDef \* huart )

Optional error handling of UART.

#### **Parameters**

huart

This is called from inside of HAL when an UART error interrupt is accepted.

STM32Cube HAL has same name function internally. That function is invoked whenever an relevant interrupt happens. In the other hand, that function is declared as weak bound. As a result, this function overrides the default error interrupt call back.

In this call back, the uart device handle have to be passed to the murasaki::Uart::HandleError() function.

52 Module Documentation

```
10.5.2.11 void HAL_UART_RxCpltCallback ( UART_HandleTypeDef * huart )
```

Essential to sync up with UART.

#### **Parameters**

```
huart
```

This is called from inside of HAL when an UART receive done interrupt is accepted.

STM32Cube HAL has same name function internally. That function is invoked whenever an relevant interrupt happens. In the other hand, that function is declared as weak bound. As a result, this function overrides the default RX interrupt call back.

In this call back, the uart device handle have to be passed to the murasaki::Uart::ReceiveCompleteCallback() function.

```
10.5.2.12 void HAL_UART_TxCpltCallback ( UART_HandleTypeDef * huart )
```

Essential to sync up with UART.

#### **Parameters**

```
huart
```

This is called from inside of HAL when an UART transmission done interrupt is accepted.

STM32Cube HAL has same name function internally. That function is invoked whenever an relevant interrupt happens. In the other hand, that function is declared as weak bound. As a result, this function overrides the default TX interrupt call back.

In this call back, the uart device handle have to be passed to the murasaki::Uart::TransmissionCompleteCallback() function.

```
10.5.2.13 void InitPlatform ( )
```

Initialize the platform variables.

The murasaki::platform variable is an interface between the application program and HAL / RTOS. To use it correctly, the initialization is needed before any activity of murasaki client.

This function have to be invoked from the StartDefaultTask() of the main.c only once to initialize the platform varaiable.

## 10.5.3 Variable Documentation

10.5.3.1 murasaki::Debugger \* murasaki::debugger

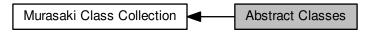
Grobal variable to provide the debugging function.

This variable is declared by murasaki platform. But not instantiated. To make it happen, programmer have to make an variable and initialize it explicitly. Otherwise, Certain debug utility/macro may cause link error, because murasaki::debugger is refered by these utility/macros.

54 Module Documentation

## 10.6 Abstract Classes

Collaboration diagram for Abstract Classes:



## **Classes**

- · class murasaki::AudioCodecStrategy
- class murasaki::BitInStrategy
- · class murasaki::BitOutStrategy
- · class murasaki::FifoStrategy
- · class murasaki::I2CMasterStrategy
- class murasaki::I2cSlaveStrategy
- · class murasaki::LoggerStrategy
- · class murasaki::PeripheralStrategy
- · class murasaki::SpiMasterStrategy
- · class murasaki::SpiSlaveSpecifierStrategy
- · class murasaki::SpiSlaveStrategy
- · class murasaki::TaskStrategy
- · class murasaki::UartStrategy

## 10.6.1 Detailed Description

Usually, application dodesn't instanciate these classes. But pointer may be clecalared as abstract class as geneic placeholder.

10.7 Helper classes 55

## 10.7 Helper classes

Collaboration diagram for Helper classes:



## Classes

- class murasaki::DebuggerFifo
- struct murasaki::LoggingHelpers
- class murasaki::DebuggerUart

#### **Functions**

- void \* operator new (std::size\_t size)
- void \* operator new[] (std::size\_t size)
- void operator delete (void \*ptr)
- void operator delete[] (void \*ptr)

## 10.7.1 Detailed Description

These classess are not used by customer.

## 10.7.2 Function Documentation

10.7.2.1 void operator delete (void \* ptr)

Deallocate the given memory.

#### **Parameters**

ptr | Pointer to the memory to deallocate

#### Returns

Allocated memory in FreeRTOS heap. Null mean fail to allocate.

56 Module Documentation

10.7.2.2 void operator delete[] ( void \* ptr )

Deallocate the given memory.

#### **Parameters**

ptr Pointer to the memory to deallocate

#### Returns

Allocated memory in FreeRTOS heap. Null mean fail to allocate.

10.7.2.3 void\* operator new ( std::size\_t size )

Allocate a memory piece with given size.

#### **Parameters**

size Size of the memory to allocate [byte]

#### Returns

Allocated memory in FreeRTOS heap. Null mean fail to allocate.

10.7.2.4 void\* operator new[] ( std::size\_t size )

Allocate a memory piece with given size.

#### **Parameters**

size Size of the memory to allocate [byte]

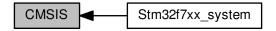
#### Returns

Allocated memory in FreeRTOS heap. Null mean fail to allocate.

10.8 CMSIS 57

## 10.8 CMSIS

Collaboration diagram for CMSIS:



## **Modules**

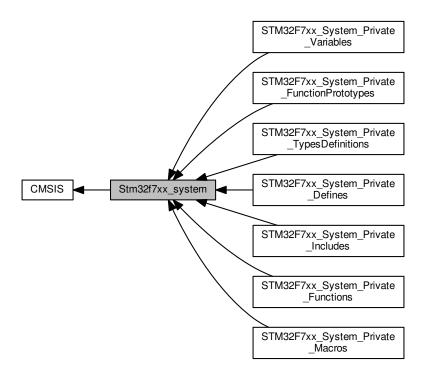
• Stm32f7xx\_system

## 10.8.1 Detailed Description

58 Module Documentation

## 10.9 Stm32f7xx\_system

Collaboration diagram for Stm32f7xx\_system:



## Modules

- STM32F7xx\_System\_Private\_Includes
- STM32F7xx\_System\_Private\_TypesDefinitions
- STM32F7xx\_System\_Private\_Defines
- STM32F7xx\_System\_Private\_Macros
- STM32F7xx\_System\_Private\_Variables
- STM32F7xx System Private FunctionPrototypes
- STM32F7xx\_System\_Private\_Functions

## 10.9.1 Detailed Description

## 10.10 STM32F7xx\_System\_Private\_Includes

Collaboration diagram for STM32F7xx\_System\_Private\_Includes:



#### **Macros**

- #define HSE\_VALUE ((uint32\_t)25000000)
- #define HSI\_VALUE ((uint32\_t)16000000)
- 10.10.1 Detailed Description
- 10.10.2 Macro Definition Documentation
- 10.10.2.1 #define HSE\_VALUE ((uint32\_t)25000000)

Default value of the External oscillator in Hz

10.10.2.2 #define HSI\_VALUE ((uint32\_t)16000000)

Value of the Internal oscillator in Hz

60 Module Documentation

# 10.11 STM32F7xx\_System\_Private\_TypesDefinitions

 $Collaboration\ diagram\ for\ STM32F7xx\_System\_Private\_TypesDefinitions:$ 



## 10.12 STM32F7xx\_System\_Private\_Defines

Collaboration diagram for STM32F7xx\_System\_Private\_Defines:



#### **Macros**

- #define VECT\_TAB\_OFFSET 0x00
- 10.12.1 Detailed Description
- 10.12.2 Macro Definition Documentation
- 10.12.2.1 #define VECT\_TAB\_OFFSET 0x00
- < Uncomment the following line if you need to relocate your vector Table in Internal SRAM. Vector Table base offset field. This value must be a multiple of 0x200.

Module Documentation

# 10.13 STM32F7xx\_System\_Private\_Macros

Collaboration diagram for STM32F7xx\_System\_Private\_Macros:



## 10.14 STM32F7xx\_System\_Private\_Variables

Collaboration diagram for STM32F7xx\_System\_Private\_Variables:



## 10.14.1 Detailed Description

Module Documentation

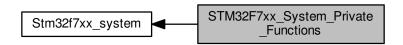
## 10.15 STM32F7xx\_System\_Private\_FunctionPrototypes

 $Collaboration\ diagram\ for\ STM32F7xx\_System\_Private\_FunctionPrototypes:$ 



## 10.16 STM32F7xx\_System\_Private\_Functions

Collaboration diagram for STM32F7xx System Private Functions:



#### **Functions**

- void SystemInit (void)
- void SystemCoreClockUpdate (void)

#### 10.16.1 Detailed Description

#### 10.16.2 Function Documentation

10.16.2.1 void SystemCoreClockUpdate (void)

Update SystemCoreClock variable according to Clock Register Values. The SystemCoreClock variable contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.

#### Note

Each time the core clock (HCLK) changes, this function must be called to update SystemCoreClock variable value. Otherwise, any configuration based on this variable will be incorrect.

- The system frequency computed by this function is not the real frequency in the chip. It is calculated based on the predefined constant and the selected clock source:
- If SYSCLK source is HSI, SystemCoreClock will contain the HSI\_VALUE(\*)
- If SYSCLK source is HSE, SystemCoreClock will contain the HSE\_VALUE(\*\*)
- If SYSCLK source is PLL, SystemCoreClock will contain the HSE\_VALUE(\*\*) or HSI\_VALUE(\*) multiplied/divided by the PLL factors.
- (\*) HSI\_VALUE is a constant defined in stm32f7xx\_hal\_conf.h file (default value 16 MHz) but the real value may vary depending on the variations in voltage and temperature.
- (\*\*) HSE\_VALUE is a constant defined in stm32f7xx\_hal\_conf.h file (default value 25 MHz), user has to ensure that HSE\_VALUE is same as the real frequency of the crystal used. Otherwise, this function may have wrong result.
  - The result of this function could be not correct when using fractional value for HSE crystal.

66	Module Documentation
Parameters	
None	
Return values	
None	
10.16.2.2 void SystemInit ( void )	
Setup the microcontroller system Initialize the Embedded Flash Interface, the PLL and up variable.	date the SystemFrequency
Parameters	
None	

Return values

None

# **Chapter 11**

# **Namespace Documentation**

## 11.1 murasaki Namespace Reference

#### Classes

- class Adau1361
- · class AudioCodecStrategy
- · class BitIn
- · class BitInStrategy
- class BitOut
- · class BitOutStrategy
- class CriticalSection
- class Debugger
- · class DebuggerFifo
- · class DebuggerUart
- · class FifoStrategy
- struct GPIO\_type
- · class I2cMaster
- class I2CMasterStrategy
- class I2cSlave
- class I2cSlaveStrategy
- class LoggerStrategy
- struct LoggingHelpers
- · class PeripheralStrategy
- struct Platform
- class SpiMaster
- class SpiMasterStrategy
- class SpiSlave
- class SpiSlaveSpecifier
- class SpiSlaveSpecifierStrategy
- class SpiSlaveStrategy
- class Synchronizer
- class Task
- class TaskStrategy
- class Uart
- · class UartLogger
- · class UartStrategy

#### **Functions**

- void SetSyslogSererityThreshold (murasaki::SyslogSeverity severity)
- void SetSyslogFacilityMask (uint32\_t mask)
- void AddSyslogFacilityToMask (murasaki::SyslogFacility facility)
- · void RemoveSyslogFacilityFromMask (murasaki::SyslogFacility facility)
- bool AllowedSyslogOut (murasaki::SyslogFacility facility, murasaki::SyslogSeverity severity)

#### **Variables**

- Debugger \* debugger
- · Platform platform

#### 11.1.1 Detailed Description

This name space encloses personal collections of the software parts to create a "platform" of the software development. This specific collection is based on the STM32Cube HAL and FreeRTOS, both are generated by CubeMX.

#### 11.1.2 Function Documentation

11.1.2.1 void murasaki::AddSyslogFacilityToMask ( murasaki::SyslogFacility facility )

Add Syslog facility to the filter mask.

#### **Parameters**

facility	Allow this facility to output

See AllowedSyslogOut to understand when the message is out.

11.1.2.2 bool murasaki::AllowedSyslogOut ( murasaki::SyslogFacility facility, murasaki::SyslogSeverity severity )

Check if given facility and severity message is allowed to output.

#### **Parameters**

facility	Message facility
severity	Message seveirty

#### Returns

True if the message is allowed to out. False if not allowed.

By comapring internal seveiry threshold and facility mask, decide whether the message can be out or not.

If seveirty is higher than or equal to kseError, message is allowed to out.

If the severity is lower than kseError, the message is allowered to out only whhen :

- The seveiry is higher than or equal to the internal threshold
- The facility is "1" in the corresponding bit of the internal facility mask.

11.1.2.3 void murasaki::RemoveSyslogFacilityFromMask ( murasaki::SyslogFacility facility )

Remove Syslog facility to the filter mask.

#### **Parameters**

facility [	Deny this facility to output
------------	------------------------------

See AllowedSyslogOut to understand when the message is out.

11.1.2.4 void murasaki::SetSyslogFacilityMask ( uint32\_t mask )

Set the syslog facility mask.

#### **Parameters**

mask	Facility bit mask. "1" allows output of the corresponding facility
------	--

The parameter is not the facility. A bit mask. By default, the bit mask is 0xFFFFFFFF which allows all facility.

See AllowedSyslogOut to understand when the message is out.

11.1.2.5 void murasaki::SetSyslogSererityThreshold ( murasaki::SyslogSeverity severity )

Set the syslog severity threshold.

#### **Parameters**

severity

Set the severity threshold. The message below this levels are ignored.

- 11.1.3 Variable Documentation
- 11.1.3.1 murasaki::Platform murasaki::platform

Grobal variable to provide the access to the platform component.

This variable is declared by murasaki platform. But not instantiated. To make it happen, programmer have to make an variable and initilize it explicitly.

Note that the instantiation of this variable is optional. This is provided just of ease of read.

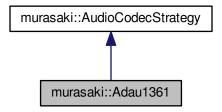
# **Chapter 12**

# **Class Documentation**

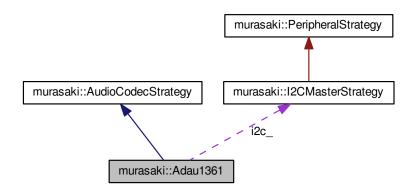
## 12.1 murasaki::Adau1361 Class Reference

#include <adau1361.hpp>

Inheritance diagram for murasaki::Adau1361:



Collaboration diagram for murasaki::Adau1361:



#### **Public Member Functions**

- Adau1361 (unsigned int fs, murasaki::I2CMasterStrategy \*controler, unsigned int i2c\_device\_addr)
- virtual void start (void)
- virtual void set line input gain (float left gain, float right gain, bool mute=false)
- virtual void set\_aux\_input\_gain (float left\_gain, float right\_gain, bool mute=false)
- virtual void set\_line\_output\_gain (float left\_gain, float right\_gain, bool mute=false)
- virtual void set\_hp\_output\_gain (float left\_gain, float right\_gain, bool mute=false)

#### **Protected Member Functions**

- virtual void configure\_pll (void)=0
- virtual void configure board (void)=0
- virtual void send\_command (const uint8\_t command[], int size)
- virtual void send\_command\_table (const uint8\_t table[][3], int rows)
- virtual void wait\_pll\_lock (void)

#### 12.1.1 Constructor & Destructor Documentation

12.1.1.1 murasaki::Adau1361::Adau1361 ( unsigned int *fs,* murasaki::I2CMasterStrategy \* *controler,* unsigned int *i2c\_device\_addr* )

constructor.

#### **Parameters**

fs	Sampling frequency.
controler	Pass the I2C controler object.
i2c_device_addr	I2C device address. value range is from 0 to 127

initialize the internal variables.

#### 12.1.2 Member Function Documentation

12.1.2.1 virtual void murasaki::Adau1361::configure\_board ( void ) [protected], [pure virtual]

configuration of the ADAU1361 for the codec board

A pure virutal function.

This member function must be overriden by inherited class. Before the calling of this function, the codec is initialized as default state except PLL. PLL is set by configure\_pll() method before calling this function.

This member funciton must configure the ADAU1361 registered based on the board circuit. For example, internal signal pass or bias.

12.1.2.2 virtual void murasaki::Adau1361::configure\_pll( void ) [protected], [pure virtual]

configuration of PLL for the desired core clock

A pure virutal function.

This member function must be overriden by inherited class. Before the call of this function, R0 is initialized as 0 and then, set the clock source is PLL.

This member funciton must configure the PLL correctly, confirm the PLL lock status. And then set the SRC.

Note that the setting SRC before PLL lock may fail.

```
12.1.2.3 virtual void murasaki::Adau1361::send_command ( const uint8_t command[], int size ) [protected], [virtual]
```

send one command to ADAU1361.

Service function for the ADAu1361 board implementer.

#### **Parameters**

command	command data array. It have to have register addess of ADAU1361 in first two bytes.
size	number of bytes in the command, including the regsiter address.

Send one complete command to ADAU3161 by I2C.

```
12.1.2.4 virtual void murasaki::Adau1361::send_command_table ( const uint8_t table[][3], int rows ) [protected], [virtual]
```

send one command to ADAU1361.

#### **Parameters**

table	command table. All commands are stored in one row. Each row has only 1 byte data after reg address.
rows	number of the rows in the table.

Service function for the ADAu1361 board implementer.

Send a list of command to ADAU1361. All commands has 3 bytes length. That mean, after two byte register address, only 1 byte data payload is allowed. Commadns are sent by I2C

```
12.1.2.5 virtual void murasaki::Adau1361::set_aux_input_gain ( float left_gain, float right_gain, bool mute = false )

[virtual]
```

Set the aux input gain and enable the relevant mixer.

#### **Parameters**

	left_gain	Gain by dB. The gain value outside of the acceptable range will be saturated.
	right_gain	Gain by dB. The gain value outside of the acceptable range will be saturated.
Ī	mute	set true to mute

Other input lines are not killed. To kill it, user have to mute them explicitly.

Reimplemented from murasaki::AudioCodecStrategy.

```
12.1.2.6 virtual void murasaki::Adau1361::set_hp_output_gain ( float left_gain, float right_gain, bool mute = false )

[virtual]
```

Set the headphone output gain and enable the relevant mixer.

#### **Parameters**

left_gain	Gain by dB. The gain value outside of the acceptable range will be saturated.
right_gain	Gain by dB. The gain value outside of the acceptable range will be saturated.
mute	set true to mute

Other out line like line in are not killed. To kill it, user have to mute them explicitly.

Reimplemented from murasaki::AudioCodecStrategy.

```
12.1.2.7 virtual void murasaki::Adau1361::set_line_input_gain ( float left_gain, float right_gain, bool mute = false )
[virtual]
```

Set the line input gain and enable the relevant mixer.

#### **Parameters**

left_gain	Gain by dB. The gain value outside of the acceptable range will be saturated.
right_gain	Gain by dB. The gain value outside of the acceptable range will be saturated.
mute	set true to mute

As same as start(), this gain control function uses the single-end negative input only. Other input signal of the line in like positive signal or diff signal are killed.

Other input line like aux are not killed. To kill it, user have to mute them explicitly.

 $Reimplemented\ from\ muras a ki:: Audio Codec Strategy.$ 

```
12.1.2.8 virtual void murasaki::Adau1361::set_line_output_gain ( float left_gain, float right_gain, bool mute = false )
[virtual]
```

Set the line output gain and enable the relevant mixer.

#### **Parameters**

left_gain	Gain by dB. The gain value outside of the acceptable range will be saturated.
right_gain	Gain by dB. The gain value outside of the acceptable range will be saturated.
mute	set true to mute

Other output lines are not killed. To kill it, user have to mute them explicitly.

Reimplemented from murasaki::AudioCodecStrategy.

12.1.2.9 virtual void murasaki::Adau1361::start (void ) [virtual]

Set up the ADAU1361 codec, and then, start the codec.

This method starts the ADAU1361 AD/DA conversion and I2S communication.

The line in is configured to use the Single-End negative input. This is funny but ADAU1361 datasheet specifies to do it. The positive in and diff in are killed. All biases are set as "normal".

The CODEC is configured as master mode. That mean, bclk and WS are given from ADAU1361 to the micro processor.

Implements murasaki::AudioCodecStrategy.

12.1.2.10 virtual void murasaki::Adau1361::wait\_pll\_lock( void ) [protected], [virtual]

wait until PLL locks.

Service function for the ADAu1361 board implementer.

Read the PLL status and repeat it until the PLL locks.

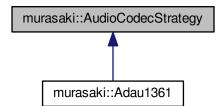
The documentation for this class was generated from the following file:

/home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc-tp/adau1361.hpp

## 12.2 murasaki::AudioCodecStrategy Class Reference

#include <audiocodecstrategy.hpp>

Inheritance diagram for murasaki::AudioCodecStrategy:



#### **Public Member Functions**

- AudioCodecStrategy (unsigned int fs)
- virtual void start (void)=0
- virtual void set line input gain (float left gain, float right gain, bool mute=false)
- virtual void set\_aux\_input\_gain (float left\_gain, float right\_gain, bool mute=false)
- virtual void set\_mic\_input\_gain (float left\_gain, float right\_gain, bool mute=false)
- virtual void set\_line\_output\_gain (float left\_gain, float right\_gain, bool mute=false)
- virtual void <a href="mailto:set\_hp\_output\_gain">set\_hp\_output\_gain</a> (float left\_gain, float right\_gain, bool mute=false)

#### 12.2.1 Detailed Description

This class is template for all codec classes

#### 12.2.2 Constructor & Destructor Documentation

12.2.2.1 murasaki::AudioCodecStrategy::AudioCodecStrategy ( unsigned int fs ) [inline]

constructor.

#### **Parameters**

```
fs Sampling frequency.
```

initialize the internal variables.

#### 12.2.3 Member Function Documentation

Set the aux input gain and enable the relevant mixer.

#### **Parameters**

left_gain	Gain by dB. The gain value outside of the acceptable range will be saturated.
right_gain	Gain by dB. The gain value outside of the acceptable range will be saturated.
mute	set true to mute

Reimplemented in murasaki::Adau1361.

12.2.3.2 virtual void murasaki::AudioCodecStrategy::set\_hp\_output\_gain ( float left\_gain, float right\_gain, bool mute = false ) [inline], [virtual]

Set the headphone output gain and enable the relevant mixer.

#### **Parameters**

left_gain	Gain by dB. The gain value outside of the acceptable range will be saturated.
right_gain	Gain by dB. The gain value outside of the acceptable range will be saturated.
mute	set true to mute

Reimplemented in murasaki::Adau1361.

Set the line input gain and enable the relevant mixer.

#### **Parameters**

left_gain	Gain by dB. The gain value outside of the acceptable range will be saturated.
right_gain	Gain by dB. The gain value outside of the acceptable range will be saturated.
mute	set true to mute

Reimplemented in murasaki::Adau1361.

12.2.3.4 virtual void murasaki::AudioCodecStrategy::set\_line\_output\_gain ( float left\_gain, float right\_gain, bool mute = false ) [inline], [virtual]

Set the line output gain and enable the relevant mixer.

#### **Parameters**

left_gain	Gain by dB. The gain value outside of the acceptable range will be saturated.
right_gain	Gain by dB. The gain value outside of the acceptable range will be saturated.
mute	set true to mute

Reimplemented in murasaki::Adau1361.

12.2.3.5 virtual void murasaki::AudioCodecStrategy::set\_mic\_input\_gain ( float left\_gain, float right\_gain, bool mute = false ) [inline], [virtual]

Set the mic input gain and enable the relevant mixer.

#### **Parameters**

left_gain	Gain by dB. The gain value outside of the acceptable range will be saturated.
right_gain	Gain by dB. The gain value outside of the acceptable range will be saturated.
mute	set true to mute

12.2.3.6 virtual void murasaki::AudioCodecStrategy::start ( void ) [pure virtual]

Actual initializer.

Initialize the codec itself and start the conversion process. and configure for given parameter.

Finally, set the input gain to 0dB.

Implemented in murasaki::Adau1361.

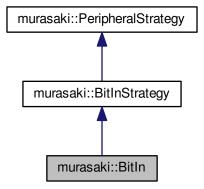
The documentation for this class was generated from the following file:

• /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/audiocodecstrategy.hpp

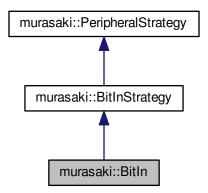
## 12.3 murasaki::BitIn Class Reference

#include <bitin.hpp>

Inheritance diagram for murasaki::BitIn:



 $Collaboration\ diagram\ for\ murasaki::BitIn:$ 



#### **Public Member Functions**

- BitIn (GPIO\_TypeDef \*port, uint16\_t pin)
- virtual unsigned int Get (void)
- virtual void \* GetPeripheralHandle ()

#### 12.3.1 Detailed Description

The BitIn class is the wrapper of the GPIO controller. To use the BitIn class, make an instance with GPIO\_TypeDef \* type pointer. For example, to create an instance for a switch peripheral:

```
my_swithc = new murasaki::BitIn(sw_port, sw_pin);
```

Where sw\_port and sw\_pin are the macro generated by CubeMX for GPIO pin. the GPIO peripheral have to be configured to be right direction.

#### 12.3.2 Constructor & Destructor Documentation

```
12.3.2.1 murasaki::Bitln::Bitln ( GPIO_TypeDef * port, uint16_t pin )
```

Constructor.

#### **Parameters**

port	Pinter to the port strict.
pin	Number of the pin to input.

#### 12.3.3 Member Function Documentation

```
12.3.3.1 unsigned int murasaki::Bitln::Get ( void ) [virtual]
```

Get a status of the output pin.

#### Returns

1 or 0 as output state.

The mean of "1" or "0" is system dependent.

Usually, these represent "H" or "L" output state, respectively.

Implements murasaki::BitInStrategy.

12.3.3.2 void \* murasaki::Bitln::GetPeripheralHandle() [virtual]

pass the raw peripheral handler

Returns

pointer to the GPIO\_type variable hidden in a class.

Implements murasaki::PeripheralStrategy.

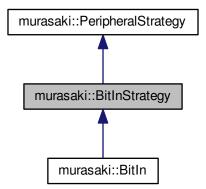
The documentation for this class was generated from the following files:

- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/bitin.hpp
- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/Src/bitin.cpp

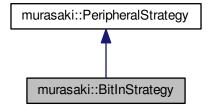
## 12.4 murasaki::BitInStrategy Class Reference

#include <bitinstrategy.hpp>

Inheritance diagram for murasaki::BitInStrategy:



Collaboration diagram for murasaki::BitInStrategy:



#### **Public Member Functions**

• virtual unsigned int Get (void)=0

## 12.4.1 Detailed Description

A prototype of the general purpose bit input class

#### 12.4.2 Member Function Documentation

12.4.2.1 virtual unsigned int murasaki::BitlnStrategy::Get ( void ) [pure virtual]

Get a status of the input pin.

Returns

1 or 0 as output state.

The mean of "1" or "0" is system dependent.

Usually, these represent "H" or "L" input state, respectively.

Implemented in murasaki::BitIn.

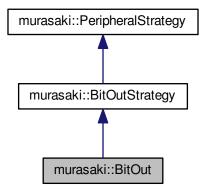
The documentation for this class was generated from the following file:

/home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/bitinstrategy.hpp

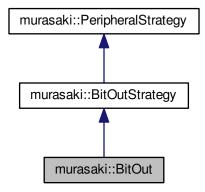
## 12.5 murasaki::BitOut Class Reference

#include <bitout.hpp>

Inheritance diagram for murasaki::BitOut:



Collaboration diagram for murasaki::BitOut:



## **Public Member Functions**

- BitOut (GPIO\_TypeDef \*port, uint16\_t pin)
- virtual void Set (unsigned int state=1)
- virtual unsigned int Get (void)
- virtual void \* GetPeripheralHandle ()

## 12.5.1 Detailed Description

The BitOut class is the wrapper of the GPIO controller. To use the BitOut class, make an instance with GPIO\_← TypeDef \* type pointer. For example, to create an instance for the a peripheral:

```
my_LED = new murasaki::BitOut(LED_port, LED_pin);
```

Where LED\_port and LED\_pin are the macro generated by CubeMX for GPIO pin. the GPIO peripheral have to be configured to be right direction.

## 12.5.2 Constructor & Destructor Documentation

12.5.2.1 murasaki::BitOut::BitOut ( GPIO\_TypeDef \* port, uint16\_t pin )

#### Constructor.

#### **Parameters**

port	Pinter to the port strict.
pin	Number of the pin to output.

## 12.5.3 Member Function Documentation

```
12.5.3.1 unsigned int murasaki::BitOut::Get(void) [virtual]
```

Get a status of the output pin.

Returns

1 or 0 as output state.

The mean of "1" or "0" is system dependent.

Usually, these represent "H" or "L" output state, respectively.

Implements murasaki::BitOutStrategy.

```
12.5.3.2 void * murasaki::BitOut::GetPeripheralHandle( ) [virtual]
```

pass the raw peripheral handler

Returns

pointer to the GPIO\_type variable hidden in a class.

Implements murasaki::PeripheralStrategy.

```
12.5.3.3 void murasaki::BitOut::Set ( unsigned int state = 1 ) [virtual]
```

Set a status of the output pin.

**Parameters** 

```
state | Set "H" if the value is none zero, vice versa.
```

Implements murasaki::BitOutStrategy.

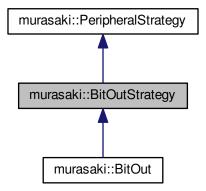
The documentation for this class was generated from the following files:

- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/Inc/bitout.hpp
- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/Src/bitout.cpp

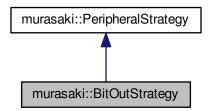
## 12.6 murasaki::BitOutStrategy Class Reference

```
#include <bitoutstrategy.hpp>
```

Inheritance diagram for murasaki::BitOutStrategy:



Collaboration diagram for murasaki::BitOutStrategy:



## **Public Member Functions**

- virtual void Set (unsigned int state=1)=0
- virtual unsigned int Get (void)=0

## 12.6.1 Detailed Description

A prototype of the general purpose bit out class

#### 12.6.2 Member Function Documentation

12.6.2.1 virtual unsigned int murasaki::BitOutStrategy::Get ( void ) [pure virtual]

Get a status of the output pin.

#### Returns

1 or 0 as output state.

The mean of "1" or "0" is system dependent.

Usually, these represent "H" or "L" output state, respectively.

Implemented in murasaki::BitOut.

12.6.2.2 virtual void murasaki::BitOutStrategy::Set ( unsigned int state = 1 ) [pure virtual]

Set a status of the output pin.

#### **Parameters**

```
state | Set "H" if the value is none zero, vice versa.
```

Implemented in murasaki::BitOut.

The documentation for this class was generated from the following file:

/home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/bitoutstrategy.hpp

#### 12.7 murasaki::CriticalSection Class Reference

```
#include <criticalsection.hpp>
```

#### **Public Member Functions**

- void Enter ()
- · void Leave ()

#### 12.7.1 Detailed Description

The critical section prevent other task to preempt that critical section. So, a task can modify the shared variable safely inside critical section.

This class provide a critical section for the task context only. This critical section is not protected from the ISR.

The critical section have to start by CriticalSection::Enter() and quit by CriticalSection::Leave().

#### 12.7.2 Member Function Documentation

12.7.2.1 void murasaki::CriticalSection::Enter ( )

Entering critical section.

Entering critical section in task context. No other task can preemptive the task inside critical section.

12.7.2.2 void murasaki::CriticalSection::Leave ( )

Leaving crititical section.

All critical seciton started by CriticalSection::Enter() have to be quit by this member function.

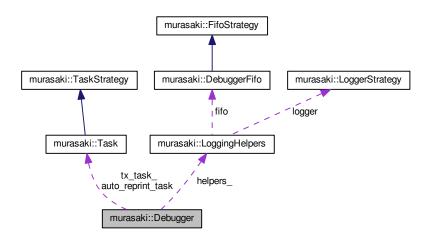
The documentation for this class was generated from the following files:

- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/Inc/criticalsection.hpp
- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/Src/criticalsection.cpp

## 12.8 murasaki::Debugger Class Reference

#include <debugger.hpp>

Collaboration diagram for murasaki::Debugger:



#### **Public Member Functions**

- Debugger (LoggerStrategy \*logger)
- void Printf (const char \*fmt,...)
- char GetchFromTask ()
- void RePrint ()
- void AutoRePrint ()

## **Protected Attributes**

- char line\_[PLATFORM\_CONFIG\_DEBUG\_LINE\_SIZE]
- · murasaki::SyslogSeverity severity\_
- uint32\_t facility\_mask\_

## 12.8.1 Detailed Description

Wrapper class to help the printf debug. The printf() method can be called from both task context and ISR context.

There are several configurable parameters of this class:

- PLATFORM\_CONFIG\_DEBUG\_BUFFER\_SIZE
- PLATFORM\_CONFIG\_DEBUG\_LINE\_SIZE
- PLATFORM CONFIG DEBUG TASK STACK SIZE
- PLATFORM CONFIG DEBUG TASK PRIORITY
- PLATFORM CONFIG DEBUG SERIAL TIMEOUT

See Application Specific Platform as example this class.

## 12.8.2 Constructor & Destructor Documentation

12.8.2.1 murasaki::Debugger::Debugger ( LoggerStrategy \* logger )

Constructor. Create internal variable.

**Parameters** 

logger The pointer to the LoggerStrategy wrapper class variable.

#### 12.8.3 Member Function Documentation

12.8.3.1 void murasaki::Debugger::AutoRePrint ( )

Print history automatically.

Once this member function is called, internally new task is created. This new task watches input by GetchFrom Task() and for each input char is recevied, trigger the RePrint().

This auto reprint function is exclusive and irreversible. Once auto reprint is triggered, there is no way to stop the auto reprint. The second call for the AutoHistory may be ignored

This member function have to be called from task context.

12.8.3.2 char murasaki::Debugger::GetchFromTask ( )

Receive one character from serial port.

Returns

Received character.

A blooking function which returns received character. The receive is done on the UART which is passed to the constructor.

This is thread safe and task context dedicated function. Never call from ISR.

Becareful, this is blocking while the Debug::Printf() non-blocking.

```
12.8.3.3 void murasaki::Debugger::Printf ( const char * fmt, ... )
```

Debug output function.

#### **Parameters**

fmt	Format string
	optional parameters

The printf() compatible method. This method can be called from both task context and ISR context. This method internally calls sprintf() variant. So, the parameter processing is fully compatible with with printf().

The formatted string is stored in the internal circular buffer. And data inside buffer is transmitted through the uart which is passed by constructor. If the buffer is overflowed, this method streos as possible, and discard the rest of string. That mean, this method is not blocking.

This member function is non-blocking, thread safe and re-entrant.

Be careful, this is non-blocking while the Debug::getchFromTask() is blocking.

At 2018/Jan/14 measurement, task stack was consumed 49bytes.

12.8.3.4 void murasaki::Debugger::RePrint ( )

Print the old data again.

Must call from task context. For each time this member function is called, old data in the buffer is re-sent again.

The data to be re-setn is the one in the data in side circular buffer. Then, the resent size is same as PLATFORM 

\_CONFIG\_DEBUG\_BUFFER\_SIZE .

## 12.8.4 Member Data Documentation

12.8.4.1 uint32\_t murasaki::Debugger::facility\_mask\_ [protected]

Syslog facility filter mask.

If certain bit is "1", the corresponding Syslog facility is allowed to output. By default the value is 0xFFFF ( equivalent to SyslogAllowAllFacilities(0xFFFFFFFF))

**12.8.4.2 char murasaki::Debugger::line\_[PLATFORM\_CONFIG\_DEBUG\_LINE\_SIZE]** [protected]

as receiver for the snprintf()

This variable can be local variable of the printf() member function. In thiss case, the implementation of the printf() is much easier. In the other hand, each task must has enough depth on its task stack.

Probably, having bigger task for each task doesn't pay, and it may cuase stack overflow bug at the debug or assertion. This is not preferable.

12.8.4.3 murasaki::SyslogSeverity murasaki::Debugger::severity\_ [protected]

Syslog severity threshold.

All seveirity level lower than this value will be ignored by Syslog() function. Note that murasaki::kseEmergency is the highest and murasaki::kseDebug is the lowerest seveirty.

By default, the severity level threshold is murasaki::kseError. That mean, the weaker severity than kseError is ignored.

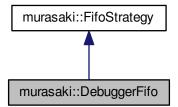
The documentation for this class was generated from the following files:

- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/debugger.hpp
- /home/takemasa/murasaki samples/nucleo-f746-sample/murasaki/Src/debugger.cpp

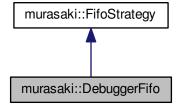
# 12.9 murasaki::DebuggerFifo Class Reference

#include <debuggerfifo.hpp>

Inheritance diagram for murasaki::DebuggerFifo:



Collaboration diagram for murasaki::DebuggerFifo:



## **Public Member Functions**

- DebuggerFifo (unsigned int buffer\_size)
- virtual unsigned int Get (uint8\_t data[], unsigned int size)
- virtual void SetPostMortem ()

## 12.9.1 Detailed Description

Non blocking, thread safe FIFO

The Put member function returns with "copied" data count. If the internal buffer is full, it returns without copy data. This is thread safe and ISR/Task bi-modal.

The Get member funciton returns with "copied" data count and data. If the internal buffer is empty, it returns without copy data.

### 12.9.2 Constructor & Destructor Documentation

12.9.2.1 murasaki::DebuggerFifo::DebuggerFifo ( unsigned int buffer\_size )

Create an internal buffer.

#### **Parameters**

Allocate the internal buffer with given buffer\_size. The buffer contents is initialized by blank.

### 12.9.3 Member Function Documentation

12.9.3.1 unsigned int murasaki::DebuggerFifo::Get ( uint8\_t data[], unsigned int size ) [virtual]

Get the data from the internal buffer. This is thread safe function. Do not call from ISR.

## **Parameters**

data	Data buffer to receive from the internal buffer
size	Size of the data parameter.

## Returns

The count of copied data. 0, if the internal buffer is empty

Reimplemented from murasaki::FifoStrategy.

12.9.3.2 void murasaki::DebuggerFifo::SetPostMortem() [virtual]

Transit to the post mortem mode.

In this mode, FIFO doesn't sync between the put and get method. Actually, this mode assumes nobody send messayge by Put()

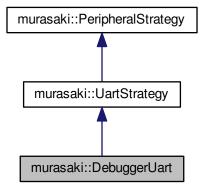
The documentation for this class was generated from the following files:

- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/debuggerfifo.hpp
- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/Src/debuggerfifo.cpp

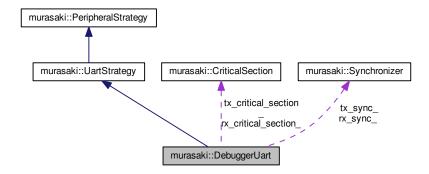
## 12.10 murasaki::DebuggerUart Class Reference

#include <debuggeruart.hpp>

Inheritance diagram for murasaki::DebuggerUart:



Collaboration diagram for murasaki::DebuggerUart:



#### **Public Member Functions**

- DebuggerUart (UART\_HandleTypeDef \*uart)
- virtual void SetHardwareFlowControl (UartHardwareFlowControl control)
- virtual void SetSpeed (unsigned int baud\_rate)
- virtual murasaki::UartStatus Transmit (const uint8\_t \*data, unsigned int size, WaitMilliSeconds timeout\_ms)
- virtual murasaki::UartStatus Receive (uint8\_t \*data, unsigned int count, unsigned int \*transfered\_count, UartTimeout uart\_timeout, WaitMilliSeconds timeout\_ms)
- virtual bool TransmitCompleteCallback (void \*const ptr)
- virtual bool ReceiveCompleteCallback (void \*const ptr)
- virtual bool HandleError (void \*const ptr)

### 12.10.1 Detailed Description

The Uart class is the wrapper of the UART controller. To use the DebuggerUart class, make an instance with UART\_HandleTypeDef \* type pointer. For example, to create an instance for the UART3 peripheral :

```
my_uart3 = new murasaki::DebuggerUart(&huart3);
```

Where huart3 is the handle generated by CubeMX for UART3 peripheral. To use this class, the UART peripheral have to be configured to use the DMA functionality. The baud rate, length and flow control should be configured by the CubeMX.

In addition to the instantiation, we need to prepare an interrupt callback.

```
void HAL_UART_TxCpltCallback(UART_HandleTypeDef * huart)
{
    my_uart3->TransmitCompleteCallback(huart);
}
```

Where HAL\_UART\_TxCpltCallback is a predefined name of the UART interrupt handler. This is invoked by system whenever a DMA baed UART transmission is complete. Becuase the default function is weakly bound, above definition will overwride the default one.

Note that above callback is invoked for any UARTn where n is 1, 2, 3... To avoid the confusion, Uart::Transmit← CompleteCallback() method chckes whether given parameter matches with its UART\_HandleTypeDef \* pointer ( which was passed to constructor ). And only when both matches, the member function execute the interrupt termination process.

As same as Tx, RX needs HAL\_UART\_TxCpltCallback().

Once the instance and callbacks are correctly prepared, we can use the Tx/Rx member function.

The Uart::Transmit() member function is a blocking function. A programmer can specify the timeout by timeout\_ms parameter. By default, this parameter is set by kwmsIndefinitely which specifes never time out.

The Uart::Receive() member function is a blocking function. A programmer can specify the timeout by timeout\_ms parameter. By default, this parameter is set by kwmsIndefinitely which specifes never time out.

Both methods can be called from only the task context. If these are called in the ISR context, the result is unknown.

## 12.10.2 Constructor & Destructor Documentation

12.10.2.1 murasaki::DebuggerUart::DebuggerUart ( UART\_HandleTypeDef \* uart )

Constructor.

### **Parameters**

uart	Pointer to a UART control struct. This device have to be configured to use DMA and interrupt for both Tx	
	and Rx.	

Store the given uart pointer into the internal variable. This pointer is passed to the STM32Cube HAL UART functions when needed.

## 12.10.3 Member Function Documentation

12.10.3.1 bool murasaki::DebuggerUart::HandleError ( void \*const ptr ) [virtual]

Error handling.

#### **Parameters**

ptr	Pointer to UART_HandleTypeDef struct.
-----	---------------------------------------

### Returns

true: ptr matches with UART device and handle the error. false : doesn't match.

A handle to print out the error message.

Checks whether handle has error and if there is, print appropriate error. Then return.

Implements murasaki::UartStrategy.

12.10.3.2 murasaki::UartStatus murasaki::DebuggerUart::Receive ( uint8\_t \* data, unsigned int count, unsigned int \* transfered\_count, UartTimeout uart\_timeout, WaitMilliSeconds timeout\_ms ) [virtual]

Receive raw data through an UART by blocking mode.

## Parameters

data	Data buffer to place the received data
count	The count of the data (byte) to be transfered. Must be smaller than 65536
transfered_count	This parameter is ignored.
uart_timeout	This parameter is ignored
timeout_ms	Time out limit by milliseconds.

## Returns

Always returns OK

Receive to given data buffer through an UART device.

The receiving mode is blocking. That means, function returns when specified number of data has been received, except timeout. Passing murasaki::kwmsIndefinitely to the parameter timeout\_ms orders not to return until complete receiving. Other value of timeout\_ms parameter specifies the time out by millisecond. If time out happen, function returns false. If not happen, it returns true.

This function is exclusive. Internally this function is guarded by mutex. Then this function is thread safe. This function is forbiddedn to call from ISR.

Implements murasaki::UartStrategy.

12.10.3.3 bool murasaki::DebuggerUart::ReceiveCompleteCallback (void \*const ptr) [virtual]

Call back for entire block transfer completion.

#### **Parameters**

ptr Pointer to UART\_HandleTypeDef struct.

#### Returns

true: ptr matches with UART device and handle the call back. false : doesn't match.

A call back to notify the end of entire block transfer. This is considered as the end of DMA based receiving. The context have to be interrupt.

This member function checks whether the given ptr parameter matches its own device, and if matched, Release the waiting task and return true. If it doesn't match, just return false.

This method have to be called from HAL\_UART\_RxCpltCallback(). See STM32F7 HAL manual for detail

Implements murasaki::UartStrategy.

12.10.3.4 void murasaki::DebuggerUart::SetHardwareFlowControl ( UartHardwareFlowControl control ) [virtual]

Set the behavior of the hardware flow control.

#### **Parameters**

control The control mode.

Before calling this method, all transmission and recevie activites have to be finished. This is responsibility of the programmer.

Note this method is NOT re-etnrant. In other word, this member function can be called from both task and interrupt context.

Reimplemented from murasaki::UartStrategy.

12.10.3.5 void murasaki::DebuggerUart::SetSpeed ( unsigned int baud\_rate ) [virtual]

Set the BAUD rate.

#### **Parameters**

baud_rate	BAUD rate ( 110, 300, 57600, )
-----------	--------------------------------

Before calling this method, all transmission and recevie activites have to be finished. This is responsibility of the programmer.

Note this method is NOT re-etnrant. In other word, this member function can be called from both task and interrupt context.

Reimplemented from murasaki::UartStrategy.

12.10.3.6 murasaki::UartStatus murasaki::DebuggerUart::Transmit ( const uint8\_t \* data, unsigned int size, WaitMilliSeconds timeout\_ms ) [virtual]

Transmit raw data through an UART by blocking mode.

#### **Parameters**

data	Data buffer to be transmitted.
size	The count of the data (byte) to be transfered. Must be smaller than 65536
timeout_ms	Time out limit by milliseconds.

#### Returns

Always returns OK

Transmit given data buffer through an UART device.

The transmission mode is blocking. That means, function returns when all data has been transmitted, except timeout. Passing murasaki::kwmsIndefinitely to the parameter timeout\_ms orders not to return until complete transmission. Other value of timeout\_ms parameter specifies the time out by millisecond. If time out happen, function returns false. If not happen, it returns true.

This function is exclusive. Internally the function is guarded by mutex. Then this function is thread safe. This function is forbiddedn to call from ISR.

Implements murasaki::UartStrategy.

12.10.3.7 bool murasaki::DebuggerUart::TransmitCompleteCallback (void \*const ptr ) [virtual]

Call back for entire block transfer completion.

## **Parameters**

Pointer to UART_HandleTypeDef struct.	ptr
---------------------------------------	-----

#### Returns

true: ptr matches with UART device and handle the call back. false : doesn't match.

A call back to notify the end of entire block transfer. This is considered as the end of DMA based transmission. The context have to be interrupt.

This member function checks whether the given ptr parameter matches its own device, and if matched, Release the waiting task and return true. If it doesn't match, just return false.

This method have to be called from HAL\_UART\_TxCpltCallback(). See STM32F7 HAL manual for detail Implements murasaki::UartStrategy.

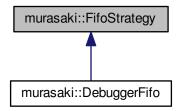
The documentation for this class was generated from the following files:

- /home/takemasa/murasaki samples/nucleo-f746-sample/murasaki/lnc/debuggeruart.hpp
- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/Src/debuggeruart.cpp

## 12.11 murasaki::FifoStrategy Class Reference

#include <fifostrategy.hpp>

Inheritance diagram for murasaki::FifoStrategy:



### **Public Member Functions**

- FifoStrategy (unsigned int buffer\_size)
- virtual unsigned int Put (uint8\_t const data[], unsigned int size)
- virtual unsigned int Get (uint8\_t data[], unsigned int size)

## 12.11.1 Detailed Description

Foundemental FIFO. No blocking, not thread safe.

The Put member function returns with "copied" data count. If the internal buffer is full, it returns without copy data.

The Get member funciton returns with "copied" data count and data. If the internal buffer is empty, it returns without copy data.

### 12.11.2 Constructor & Destructor Documentation

12.11.2.1 murasaki::FifoStrategy::FifoStrategy ( unsigned int buffer\_size )

Create an internal buffer.

#### **Parameters**

buffer_size   Size of the internal buffer to be allo	ocated [byte]
--	---------------

Allocate the internal buffer with given buffer\_size. The contents is not initialized.

### 12.11.3 Member Function Documentation

12.11.3.1 unsigned int murasaki::FifoStrategy::Get ( uint8\_t data[], unsigned int size ) [virtual]

Get the data from the internal buffer.

#### **Parameters**

data	Data buffer to receive from the internal buffer
size	Size of the data parameter.

## Returns

The count of copied data. 0, if the internal buffer is empty

Reimplemented in murasaki::DebuggerFifo.

12.11.3.2 unsigned int murasaki::FifoStrategy::Put ( uint8\_t const data[], unsigned int size ) [virtual]

Put the data into the internal buffer.

#### **Parameters**

data	Data to be copied to the internal buffer
size	Data count to be copied

#### Returns

The count of copied data. 0, if the internal buffer is full.

The documentation for this class was generated from the following files:

- $\bullet \ \ / home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/fifostrategy.hpp$
- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/Src/fifostrategy.cpp

# 12.12 murasaki::GPIO\_type Struct Reference

#include <bitout.hpp>

## 12.12.1 Detailed Description

This struct is used in the BitIn class and BitOut class. These classes returns a pointer to the variable of this type, as return value of the GetPeripheralHandle() member function.

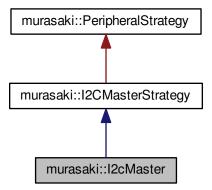
The documentation for this struct was generated from the following file:

/home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/bitout.hpp

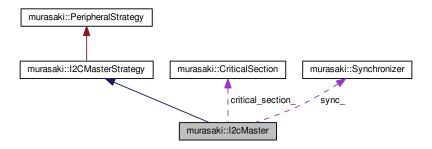
## 12.13 murasaki::I2cMaster Class Reference

#include <i2cmaster.hpp>

Inheritance diagram for murasaki::I2cMaster:



Collaboration diagram for murasaki::I2cMaster:



#### **Public Member Functions**

- I2cMaster (I2C HandleTypeDef \*i2c handle)
- virtual murasaki::|2cStatus Transmit (uint addrs, const uint8\_t \*tx\_data, unsigned int tx\_size, uint \*transfered count, WaitMilliSeconds timeout ms)
- virtual murasaki::l2cStatus Receive (uint addrs, uint8\_t \*rx\_data, unsigned int rx\_size, uint \*transfered\_count, WaitMilliSeconds timeout\_ms)
- virtual murasaki::l2cStatus TransmitThenReceive (uint addrs, const uint8\_t \*tx\_data, unsigned int tx\_size, uint8\_t \*rx\_data, unsigned int rx\_size, uint \*tx\_transfered\_count, uint \*rx\_transfered\_count, WaitMilliSeconds timeout ms)
- virtual bool TransmitCompleteCallback (void \*ptr)
- virtual bool ReceiveCompleteCallback (void \*ptr)
- virtual bool HandleError (void \*ptr)

## 12.13.1 Detailed Description

The I2cMaster class is the wrapper of the I2C controller. To use the I2cMaster class, make an instance with I2C $_{\leftarrow}$  HandleTypeDef \* type pointer. For example, to create an instance for the I2C3 peripheral :

```
my_i2c3 = new murasaki::I2cMaster(&hi2c3);
```

Where hi2c3 is the handle generated by CubeMX for I2C3 peripheral. To use this class, the I2C peripheral have to be configured to use the interrupt functionality without DMA. The bitrate should be configured by the CubeMX.

In addition to the instantiation, we need to prepare an interrupt callback.

```
void HAL_I2C_TxCpltCallback(I2C_HandleTypeDef * hi2c)
{
    my_i2c3->TransmitCompleteCallback(hi2c);
}
```

Where HAL\_I2C\_TxCpltCallback is a predefined name of the I2C interrupt handler. This is invoked by system whenever a interrupt baed I2C transmission is complete. Becuase the default function is weakly bound, above definition will overwride the default one.

Note that above callback is invoked for any I2Cn where n is 1, 2, 3... To avoid the confusion, I2cMaster::Transmit← CompleteCallback() method chckes whether given parameter matches with its I2C\_HandleTypeDef \* pointer ( which was passed to constructor ). And only when both matches, the member function execute the interrupt termination process.

As same as Tx, RX needs HAL\_I2C\_TxCpltCallback().

Once the instance and callback are correctly prepared, we can use the Tx/Rx member function.

The I2cMaster::Transmit() member function is a blocking function. A programmer can specify the timeout by timeout\_ms parameter. By default, this parameter is set by kwmsIndefinitely which specifes never time out.

The I2cMaster::Receive() member function is a blocking function. A programmer can specify the timeout by timeout\_ms parameter. By default, this parameter is set by kwmsIndefinitely which species never time out.

The I2cMaster::TransmitThenReceive() member function is blocking function. A programmer can specify the timeout by timeout\_ms parameter. By default, this parameter is set by kwmsIndefinitely which species never time out.

Both methods can be called from only the task context. If these are called in the ISR context, the result is unknown.

Note: In case an time out occurs during transmit / receive, this implementation calls HAL\_I2C\_MASTER\_ABOR ← T\_IT(). But it is unknown whether this is right thing to do. The HAL reference of the STM32F7 is not clear for this case. For example, it doesn't tell what programmer do to stop the transfer at the middle. And also, it doesn't tell what's happen if the HAL\_I2C\_MASTER\_ABORT\_IT() is called.

According to the source code of the HAL\_I2C\_MASTER\_ABORT\_IT(), no interrupt will be raised by this API call.

## 12.13.2 Constructor & Destructor Documentation

12.13.2.1 murasaki::l2cMaster::l2cMaster (  $l2C_HandleTypeDef*i2c_handle$  )

Constructor.

#### **Parameters**

i2c_handle	Peripheral handle created by CubeMx	
------------	-------------------------------------	--

### 12.13.3 Member Function Documentation

12.13.3.1 bool murasaki::l2cMaster::HandleError(void\*ptr) [virtual]

Error handling.

### **Parameters**

nter to I2C_HandleTypeDef struct.	ptr
-----------------------------------	-----

#### Returns

true: ptr matches with device and handle the error. false : doesn't match.

A handle to print out the error message.

Checks whether handle has error and if there is, print appropriate error. Then return.

Implements murasaki::I2CMasterStrategy.

12.13.3.2 murasaki::l2cStatus murasaki::l2cMaster::Receive ( uint addrs, uint8\_t \* rx\_data, unsigned int rx\_size, uint \* transfered\_count, WaitMilliSeconds timeout\_ms ) [virtual]

Thread safe, blocking receiving over I2C.

### **Parameters**

addrs	7bit address of the I2C device.
rx_data	Data array to transmit.
rx_size	Data counts[bytes] to transmit. Must be smaller than 65536
transfered_count	( Currently, Just ignored) the count of the bytes transfered during the API execution.
timeout_ms	Time ou [mS]. By default, there is not timeout.

### Returns

Result of the processing

This member function is programmed to run in the task context of RTOS. This should be internally exclusive between multiple task access. In other word, it should be thread save.

Following are the return code:

• murasaki::ki2csOK : All Receive completed.

- murasaki::ki2csNak : Receive terminated by NAK receiving.
- murasaki::ki2csArbitrationLost: Receive terminated by an arbitration error of the multi-master.
- murasaki::ki2csBussError : Receive terminated by bus error
- murasaki::ki2csTimeOut : Receive abort by timeout.
- other value: Unhandled error. I2C device are re-initialized.

Implements murasaki::I2CMasterStrategy.

12.13.3.3 bool murasaki::l2cMaster::ReceiveCompleteCallback (void \* ptr ) [virtual]

Call back to be called for entire block transfer is complete.

#### **Parameters**

ptr	Pointer for generic use. Usually, points a struct of a peripheral control
-----	---

### Returns

true: ptr matches with peripheral and handle the call back. false : doesn't match.

A call back to notify the end of entire block or byte transfer. The definition of calling timing is depend on the implementation. This is called from an DMA ISR.

Typically, an implementation may check whether the given ptr parameter matches its own device, and if matched, handle it and return true. If it doesn't match, just return false.

Implements murasaki::I2CMasterStrategy.

12.13.3.4 murasaki::l2cStatus murasaki::l2cMaster::Transmit ( uint addrs, const uint8\_t \* tx\_data, unsigned int tx\_size, uint \* transfered\_count, WaitMilliSeconds timeout\_ms ) [virtual]

Thread safe, blocking transmission over I2C.

#### **Parameters**

addrs	7bit address of the I2C device.	
tx_data	Data array to transmit.	
tx_size	Data counts[bytes] to transmit. Must be smaller than 65536	
transfered_count	( Currently, Just ignored) the count of the bytes transfered during the API execution.	
timeout_ms	Time ou [mS]. By default, there is not timeout.	

## Returns

Result of the processing

This member function is programmed to run in the task context of RTOS. This should be internally exclusive between multiple task access. In other word, it should be thread save.

Following are the return code:

- murasaki::ki2csOK : All transmission completed.
- murasaki::ki2csNak : Transmission terminated by NAK receiving.
- murasaki::ki2csArbitrationLost: Transmission terminated by an arbitration error of the multi-master.
- · murasaki::ki2csBussError: Transmission terminated by bus error
- murasaki::ki2csTimeOut : Transmission abort by timeout.
- · other value : Unhandled error. I2C device are re-initialized.

Implements murasaki::I2CMasterStrategy.

12.13.3.5 bool murasaki::l2cMaster::TransmitCompleteCallback(void \* ptr) [virtual]

Call back to be called notify the transfer is complete.

#### **Parameters**

ointer for generic use. Usually, points a struct of a peripheral control	ptr
--	-----

### Returns

true: ptr matches with peripheral and handle the call back. false : doesn't match.

A call back to notify the end of entire block or byte transfer. The definition of calling timing is depend on the implementation. This is called from an DMA ISR.

Typically, an implementation may check whether the given ptr parameter matches its own device, and if matched, handle it and return true. If it doesn't match, just return false.

Implements murasaki::I2CMasterStrategy.

12.13.3.6 murasaki::l2cStatus murasaki::l2cMaster::TransmitThenReceive ( uint addrs, const uint8\_t \* tx\_data, unsigned int tx\_size, uint8\_t \* rx\_data, unsigned int rx\_size, uint \* tx\_transfered\_count, uint \* rx\_transfered\_count, WaitMilliSeconds timeout\_ms ) [virtual]

Thread safe, blocking transmission and then receiving over I2C.

### **Parameters**

addrs	7bit address of the I2C device.
tx_data	Data array to transmit.
tx_size	Data counts[bytes] to transmit. Must be smaller than 65536
rx_data	Data array to transmit.
rx_size	Data counts[bytes] to transmit. Must be smaller than 65536
tx_transfered_count	( Currently, Just ignored) the count of the bytes transmitted during the API execution.
rx_transfered_count	( Currently, Just ignored) the count of the bytes received during the API execution.
timeout_ms	Time ou [mS]. By default, there is not timeout.

#### Returns

Result of the processing

First, this member function transmit the data, and the, by repeated start function, it receives data. The transmission device address and receiving device address is same.

This member function is programmed to run in the task context of RTOS. This should be internally exclusive between multiple task access. In other word, it should be thread save.

Following are the return code:

- murasaki::ki2csOK: All transmission and receive completed.
- murasaki::ki2csNak : Transmission or receive terminated by NAK receiving.
- murasaki::ki2csArbitrationLost: Transmission or receive terminated by an arbitration error of the multi-master.
- · murasaki::ki2csBussError: Transmission or receive terminated by bus error
- murasaki::ki2csTimeOut : Transmission or receive abort by timeout.
- other value: Unhandled error. I2C device are re-initialized.

Implements murasaki::I2CMasterStrategy.

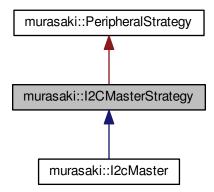
The documentation for this class was generated from the following files:

- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/i2cmaster.hpp
- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/Src/i2cmaster.cpp

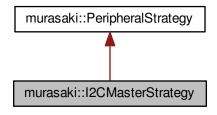
# 12.14 murasaki::I2CMasterStrategy Class Reference

#include <i2cmasterstrategy.hpp>

Inheritance diagram for murasaki::I2CMasterStrategy:



Collaboration diagram for murasaki::I2CMasterStrategy:



### **Public Member Functions**

- virtual murasaki::I2cStatus Transmit (uint addrs, const uint8\_t \*tx\_data, unsigned int tx\_size, uint \*transfered count=nullptr, WaitMilliSeconds timeout ms=murasaki::kwmsIndefinitely)=0
- virtual murasaki::l2cStatus Receive (uint addrs, uint8\_t \*rx\_data, unsigned int rx\_size, uint \*transfered\_← count=nullptr, WaitMilliSeconds timeout ms=murasaki::kwmsIndefinitely)=0
- virtual murasaki::l2cStatus TransmitThenReceive (uint addrs, const uint8\_t \*tx\_data, unsigned int tx\_size, uint8\_t \*rx\_data, unsigned int rx\_size, uint \*tx\_transfered\_count=nullptr, uint \*rx\_transfered\_count=nullptr, WaitMilliSeconds timeout\_ms=murasaki::kwmsIndefinitely)=0
- virtual bool TransmitCompleteCallback (void \*ptr)=0
- virtual bool ReceiveCompleteCallback (void \*ptr)=0
- virtual bool HandleError (void \*ptr)=0

## 12.14.1 Detailed Description

A prototype of the I2C master peripheral.

This prototype assumes the derived class will transmit / receive data in the task context on RTOS. And these member functions should be blocking. That mean, until the transmit / receive terminates, both method doesn't return.

Two call back member functions are prepared to sync with the interrupt which tells the end of Transmit/Receive.

#### 12.14.2 Member Function Documentation

12.14.2.1 virtual bool murasaki::l2CMasterStrategy::HandleError ( void \* ptr ) [pure virtual]

Handling error report of device.

#### **Parameters**

ptr Pointer for generic use. Usually, points a struct of a device control

#### Returns

true if ptr matches with device and handle the error. false if ptr doesn't match A member function to detect

The error handling is depend on the implementation.

Implemented in murasaki::I2cMaster.

```
12.14.2.2 virtual murasaki::l2cStatus murasaki::l2cMasterStrategy::Receive ( uint addrs, uint8_t * rx_data, unsigned int rx_size, uint * transfered_count = nullptr, WaitMilliSeconds timeout_ms = murasaki::kwmsIndefinitely
) [pure virtual]
```

Thread safe, blocking receiving over I2C.

#### **Parameters**

addrs	7bit address of the I2C device.
rx_data	Data array to transmit.
rx_size	Data counts[bytes] to transmit.
transfered_count	the count of the bytes transfered during the API execution.
timeout_ms	Time ou [mS]. By default, there is not timeout.

## Returns

Result of the processing

This member function is programmed to run in the task context of RTOS. This should be internally exclusive between multiple task access. In other word, it should be thread save.

Implemented in murasaki::I2cMaster.

12.14.2.3 virtual bool murasaki::l2CMasterStrategy::ReceiveCompleteCallback (void \* ptr ) [pure virtual]

Call back to be called for entire block transfer is complete.

## Parameters

ptr	Pointer for generic use. Usually, points a struct of a peripheral control
-----	---

### Returns

true: ptr matches with peripheral and handle the call back. false: doesn't match.

A call back to notify the end of entire block or byte transfer. The definition of calling timing is depend on the implementation. This is called from an DMA ISR.

Typically, an implementation may check whether the given ptr parameter matches its own device, and if matched, handle it and return true. If it doesn't match, just return false.

Implemented in murasaki::I2cMaster.

12.14.2.4 virtual murasaki::l2cStatus murasaki::l2cMasterStrategy::Transmit ( uint addrs, const uint8\_t \* tx\_data, unsigned int tx\_size, uint \* transfered\_count = nullptr, WaitMilliSeconds timeout\_ms = murasaki::kwmsIndefinitely ) [pure virtual]

Thread safe, blocking transmission over I2C.

#### **Parameters**

addrs	7bit address of the I2C device.
tx_data Data array to transmit.	
tx_size	Data counts[bytes] to transmit.
transfered_count	the count of the bytes transfered during the API execution.
timeout_ms	Time ou [mS]. By default, there is not timeout.

#### Returns

Result of the processing

This member function is programmed to run in the task context of RTOS. This should be internally exclusive between multiple task access. In other word, it should be thread save.

Implemented in murasaki::I2cMaster.

12.14.2.5 virtual bool murasaki::l2CMasterStrategy::TransmitCompleteCallback(void\*ptr) [pure virtual]

Call back to be called notify the transfer is complete.

### **Parameters**

ptr	Pointer for generic use.	Usually, points a struct of a peripheral control
-----	--------------------------	--

## Returns

true: ptr matches with peripheral and handle the call back. false: doesn't match.

A call back to notify the end of entire block or byte transfer. The definition of calling timing is depend on the implementation. This is called from an DMA ISR.

Typically, an implementation may check whether the given ptr parameter matches its own device, and if matched, handle it and return true. If it doesn't match, just return false.

Implemented in murasaki::I2cMaster.

12.14.2.6 virtual murasaki::l2cStatus murasaki::l2cMasterStrategy::TransmitThenReceive ( uint addrs, const uint8\_t \* tx\_data, unsigned int tx\_size, uint8\_t \* rx\_data, unsigned int rx\_size, uint \* tx\_transfered\_count = nullptr, uint \* rx\_transfered\_count = nullptr, WaitMilliSeconds timeout\_ms = murasaki::kwmsIndefinitely )

[pure virtual]

Thread safe, blocking transmission and then receiving over I2C.

## **Parameters**

addrs	7bit address of the I2C device.
tx_data	Data array to transmit.
tx_size	Data counts[bytes] to transmit.
rx_data	Data array to transmit.
rx_size	Data counts[bytes] to transmit.
tx_transfered_count	the count of the bytes transmitted during the API execution.
rx_transfered_count	the count of the bytes received during the API execution.
timeout_ms Time ou [mS]. By default, there is not timeout.	

### Returns

Result of the processing

First, this member function transmit the data, and the, by repeated start function, it receives data. The transmission device address and receiving device address is same.

This member function is programmed to run in the task context of RTOS. This should be internally exclusive between multiple task access. In other word, it should be thread save.

Implemented in murasaki::I2cMaster.

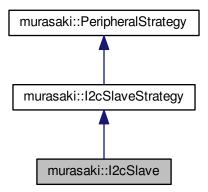
The documentation for this class was generated from the following file:

/home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/i2cmasterstrategy.hpp

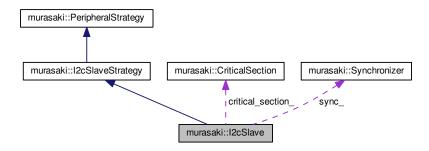
## 12.15 murasaki:: I2cSlave Class Reference

#include <i2cslave.hpp>

Inheritance diagram for murasaki::I2cSlave:



Collaboration diagram for murasaki::12cSlave:



## **Public Member Functions**

- virtual murasaki::l2cStatus Transmit (const uint8\_t \*tx\_data, unsigned int tx\_size, uint \*transfered\_count, WaitMilliSeconds timeout\_ms)
- virtual murasaki::I2cStatus Receive (uint8\_t \*rx\_data, unsigned int rx\_size, uint \*transfered\_count, Wait←
   MilliSeconds timeout\_ms)
- virtual bool TransmitCompleteCallback (void \*ptr)
- virtual bool ReceiveCompleteCallback (void \*ptr)
- virtual bool HandleError (void \*ptr)

## 12.15.1 Detailed Description

The I2cSlave class is the wrapper of the I2C controller. To use the I2cSlave class, make an instance with I2C\_ 
HandleTypeDef \* type pointer. For example, to create an instance for the I2C3 peripheral :

```
my_i2c3 = new murasaki::I2cSlave(&hi2c3);
```

Where hi2c3 is the handle generated by CubeMX for I2C3 peripheral. To use this class, the I2C peripheral have to be configured to use the interrupt functionality without DMA. The bit rate and the peripheral address should be configured by the CubeMX.

In addition to the instantiation, we need to prepare an interrupt callback. and error callback

```
void HAL_I2C_TxCpltCallback(I2C_HandleTypeDef * hi2c)
{
   if ( my_i2c3->TransmitCompleteCallback(hi2c))
      return;
}

void HAL_I2C_ErrorCallback(I2C_HandleTypeDef * hi2c)
{
   if (my_i2c3->HandleError(hi2c))
      return;
}
```

Where HAL\_I2C\_TxCpltCallback is a predefined name of the I2C interrupt handler. This is invoked by system whenever a interrupt baed I2C transmission is complete. Because the default function is weakly bound, above definition will override the default one.

Note that above callback are invoked for any I2Cn where n is 1, 2, 3... To avoid the confusion, I2cMaster::Transmit← CompleteCallback() method checks whether given parameter matches with its I2C\_HandleTypeDef \* pointer ( which was passed to constructor ). And only when both matches, the member function execute the interrupt termination process. In case of the successful match, it returns true.

As same as Tx, RX needs HAL\_I2C\_TxCpltCallback().

Once the instance and callbacks are correctly prepared, we can use the Tx/Rx member function.

The l2cSlave::Transmit() member function is a blocking function. A programmer can specify the timeout by timeout ← ms parameter. By default, this parameter is set by kwmsIndefinitely which specifes never time out.

The l2cSlave::Receive() member function is a blocking function. A programmer can specify the timeout by timeout ← ms parameter. By default, this parameter is set by kwmsIndefinitely which specifes never time out.

Both methods can be called from only the task context. If these are called in the ISR context, the result is unknown.

• Note: In case an time out occurs during transmit / receive, this implementation calls HAL\_I2C\_Delnit()/H ← AL\_I2C\_Init(). But it is unknown whether this is right thing to do. The HAL reference of the STM32F7 is not clear for this case. For example, it doesn't tell what programmer do to stop the transfer at the middle.

#### 12.15.2 Member Function Documentation

**12.15.2.1** bool murasaki::l2cSlave::HandleError(void\*ptr) [virtual]

Error handling.

**Parameters** 

```
ptr | Pointer to I2C_HandleTypeDef struct.
```

### Returns

true: ptr matches with device and handle the error. false: doesn't match.

A handle to print out the error message.

Checks whether handle has error and if there is, print appropriate error. Then return.

Implements murasaki::I2cSlaveStrategy.

12.15.2.2 murasaki::l2cStatus murasaki::l2cSlave::Receive ( uint8\_t \* rx\_data, unsigned int rx\_size, uint \* transfered\_count, WaitMilliSeconds timeout\_ms ) [virtual]

Thread safe, blocking receiving over I2C.

#### **Parameters**

rx_data	Data array to transmit.	
rx_size	Data counts[bytes] to transmit. Must be smaller than 65536	
transfered_count	( Currently, Just ignored) the count of the bytes transfered during the API execution.	
timeout_ms	Time ou [mS]. By default, there is not timeout.	

#### Returns

Result of the processing

This member function is programmed to run in the task context of RTOS. This should be internally exclusive between multiple task access. In other word, it should be thread save.

Following are the return code:

- murasaki::ki2csOK : All Receive completed.
- murasaki::ki2csNak : Receive terminated by NAK receiving.
- murasaki::ki2csArbitrationLost: Receive terminated by an arbitration error of the multi-master.
- murasaki::ki2csBussError: Receive terminated by bus error
- murasaki::ki2csTimeOut : Receive abort by timeout.
- other value : Unhandled error. I2C device are re-initialized.

Implements murasaki::I2cSlaveStrategy.

12.15.2.3 bool murasaki::l2cSlave::ReceiveCompleteCallback(void \* ptr) [virtual]

Call back to be called for entire block transfer is complete.

## **Parameters**

ptr Pointer for generic use. Usually, points a struct of a peripheral control

### Returns

true: ptr matches with peripheral and handle the call back. false : doesn't match.

A call back to notify the end of entire block or byte transfer. The definition of calling timing is depend on the implementation. This is called from an DMA ISR.

Typically, an implementation may check whether the given ptr parameter matches its own device, and if matched, handle it and return true. If it doesn't match, just return false.

Implements murasaki::I2cSlaveStrategy.

12.15.2.4 murasaki::l2cStatus murasaki::l2cSlave::Transmit ( const uint8\_t \* tx\_data, unsigned int tx\_size, uint \* transfered\_count, WaitMilliSeconds timeout\_ms ) [virtual]

Thread safe, blocking transmission over I2C.

#### **Parameters**

tx_data	Data array to transmit.	
tx_size	Data counts[bytes] to transmit. Must be smaller than 65536	
transfered_count	( Currently, Just ignored) the count of the bytes transfered during the API execution.	
timeout_ms	Time ou [mS]. By default, there is not timeout.	

#### Returns

Result of the processing

This member function is programmed to run in the task context of RTOS. This should be internally exclusive between multiple task access. In other word, it should be thread save.

Following are the return code:

- murasaki::ki2csOK : All transmission completed.
- murasaki::ki2csNak : Transmission terminated by NAK receiving.
- murasaki::ki2csArbitrationLost: Transmission terminated by an arbitration error of the multi-master.
- murasaki::ki2csBussError: Transmission terminated by bus error
- murasaki::ki2csTimeOut : Transmission abort by timeout.
- other value : Unhandled error. I2C device are re-initialized.

Implements murasaki::I2cSlaveStrategy.

12.15.2.5 bool murasaki::l2cSlave::TransmitCompleteCallback (void \* ptr ) [virtual]

Call back to be called notify the transfer is complete.

## **Parameters**

ptr Pointer for generic use. Usually, points a struct of a peripheral control

#### Returns

true: ptr matches with peripheral and handle the call back. false : doesn't match.

A call back to notify the end of entire block or byte transfer. The definition of calling timing is depend on the implementation. This is called from an DMA ISR.

Typically, an implementation may check whether the given ptr parameter matches its own device, and if matched, handle it and return true. If it doesn't match, just return false.

Implements murasaki::I2cSlaveStrategy.

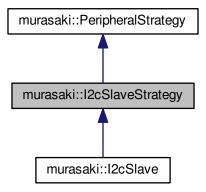
The documentation for this class was generated from the following files:

- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/Inc/i2cslave.hpp
- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/Src/i2cslave.cpp

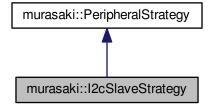
# 12.16 murasaki:: I2cSlaveStrategy Class Reference

#include <i2cslavestrategy.hpp>

Inheritance diagram for murasaki::I2cSlaveStrategy:



Collaboration diagram for murasaki::I2cSlaveStrategy:



## **Public Member Functions**

- virtual murasaki::I2cStatus Transmit (const uint8\_t \*tx\_data, unsigned int tx\_size, uint \*transfered\_←
   count=nullptr, murasaki::WaitMilliSeconds timeout\_ms=murasaki::kwmsIndefinitely)=0
- virtual murasaki::l2cStatus Receive (uint8\_t \*rx\_data, unsigned int rx\_size, uint \*transfered\_count=nullptr, murasaki::WaitMilliSeconds timeout\_ms=murasaki::kwmsIndefinitely)=0
- virtual bool TransmitCompleteCallback (void \*ptr)=0
- virtual bool ReceiveCompleteCallback (void \*ptr)=0
- virtual bool HandleError (void \*ptr)=0

## 12.16.1 Detailed Description

A prototype of the I2C slave peripheral.

This prototype assumes the derived class will transmit / receive data in the task context on RTOS. And these member functions should be blocking. That mean, until the transmit / receive terminates, both method doesn't return.

Two call back member functions are prepared to sync with the interrupt which tells the end of Transmit/Receive.

### 12.16.2 Member Function Documentation

**12.16.2.1** virtual bool murasaki::l2cSlaveStrategy::HandleError ( void \* ptr ) [pure virtual]

Handling error report of device.

#### **Parameters**

#### Returns

true if ptr matches with device and handle the error. false if ptr doesn't match A member function to detect error.

The error handling is depend on the implementation.

Implemented in murasaki::I2cSlave.

12.16.2.2 virtual murasaki::l2cStatus murasaki::l2cSlaveStrategy::Receive ( uint8\_t \* rx\_data, unsigned int rx\_size, uint \* transfered\_count = nullptr, murasaki::WaitMilliSeconds timeout\_ms = murasaki::kwmsIndefinitely )

[pure virtual]

Thread safe, blocking receiving over I2C.

#### **Parameters**

rx_data	Data array to transmit.
rx_size	Data counts[bytes] to transmit.
transfered_count	the count of the bytes transfered during the API execution.
timeout_ms	Time ou [mS]. By default, there is not timeout.

## Returns

Result of the processing

This member function is programmed to run in the task context of RTOS. This should be internally exclusive between multiple task access. In other word, it should be thread save.

Implemented in murasaki::I2cSlave.

12.16.2.3 virtual bool murasaki::l2cSlaveStrategy::ReceiveCompleteCallback(void\*ptr) [pure virtual]

Call back to be called for entire block transfer is complete.

#### **Parameters**

ptr	Pointer for generic use.	Usually, points a struct of a peripheral control
-----	--------------------------	--

#### Returns

true: ptr matches with peripheral and handle the call back. false : doesn't match.

A call back to notify the end of entire block or byte transfer. The definition of calling timing is depend on the implementation. This is called from an DMA ISR.

Typically, an implementation may check whether the given ptr parameter matches its own device, and if matched, handle it and return true. If it doesn't match, just return false.

Implemented in murasaki::I2cSlave.

12.16.2.4 virtual murasaki::l2cStatus murasaki::l2cSlaveStrategy::Transmit ( const uint8\_t \* tx\_data, unsigned int tx\_size, uint \* transfered\_count = nullptr, murasaki::WaitMilliSeconds timeout\_ms = murasaki::kwmsIndefinitely ) [pure virtual]

Thread safe, blocking transmission over I2C.

### **Parameters**

tx_data	Data array to transmit.
tx_size	Data counts[bytes] to transmit.
transfered_count	the count of the bytes transfered during the API execution.
timeout_ms	Time ou [mS]. By default, there is not timeout.

## Returns

Result of the processing

This member function is programmed to run in the task context of RTOS. This should be internally exclusive between multiple task access. In other word, it should be thread save.

Implemented in murasaki::I2cSlave.

12.16.2.5 virtual bool murasaki::l2cSlaveStrategy::TransmitCompleteCallback(void \* ptr) [pure virtual]

Call back to be called notify the transfer is complete.

## **Parameters**

ptr Pointer for generic use. Usually, points a struct of a peripheral cont
--

#### Returns

true: ptr matches with peripheral and handle the call back. false : doesn't match.

A call back to notify the end of entire block or byte transfer. The definition of calling timing is depend on the implementation. This is called from an DMA ISR.

Typically, an implementation may check whether the given ptr parameter matches its own device, and if matched, handle it and return true. If it doesn't match, just return false.

Implemented in murasaki::I2cSlave.

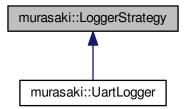
The documentation for this class was generated from the following file:

/home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/Inc/i2cslavestrategy.hpp

# 12.17 murasaki::LoggerStrategy Class Reference

#include <loggerstrategy.hpp>

Inheritance diagram for murasaki::LoggerStrategy:



## **Public Member Functions**

- virtual ~LoggerStrategy ()
- virtual void putMessage (char message[], unsigned int size)=0
- virtual char getCharacter ()=0
- virtual void DoPostMortem (void \*debugger\_fifo)

### 12.17.1 Detailed Description

A generic class to serve a logging function. This class is designed to pass to the murasaki::Debugger.

As a service class to Debug. This class's two member functions ( putMessage() and getCharacter() ) have to be able to run in the task context. Both member functions also have to be the blocking function.

## 12.17.2 Constructor & Destructor Documentation

12.17.2.1 virtual murasaki::LoggerStrategy::~LoggerStrategy( ) [inline], [virtual]

Detructor.

Do nothing here. Declared to enforce the derived class's constructor as "virtual".

#### 12.17.3 Member Function Documentation

12.17.3.1 virtual void murasaki::LoggerStrategy::DoPostMortem(void\*debugger\_fifo) [inline], [virtual]

Start post mortem process.

#### **Parameters**

debugger_fifo	Pointer to the DebuggerFifo class object. This is declared as void to avoid the include	1
	confusion. This member function read the data in given FIFO, and then do the auto history.	

By default this is not implemented. But in case user implments a method, it should call the Debugger::SetPost ← Mortem() internaly.

Reimplemented in murasaki::UartLogger.

12.17.3.2 virtual char murasaki::LoggerStrategy::getCharacter() [pure virtual]

Character input member function.

#### Returns

A character from input is returned.

This function is considered as blocking. That mean, the function will wait for any user input forever.

Implemented in murasaki::UartLogger.

12.17.3.3 virtual void murasaki::LoggerStrategy::putMessage ( char message[], unsigned int size ) [pure virtual]

Message output member function.

## Parameters

message	Non null terminated character array. This data is stored or output to the logger.
size	Byte length of the message parameter of the putMessage member function.

This function is considered as blooking. That mean, it will not wayt until data is stored to the storage or output.

Implemented in murasaki::UartLogger.

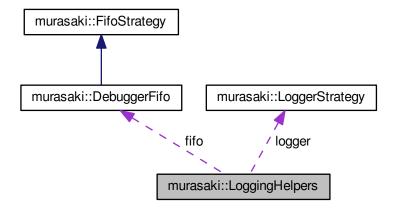
The documentation for this class was generated from the following file:

/home/takemasa/murasaki samples/nucleo-f746-sample/murasaki/lnc/loggerstrategy.hpp

# 12.18 murasaki::LoggingHelpers Struct Reference

```
#include <debuggerfifo.hpp>
```

Collaboration diagram for murasaki::LoggingHelpers:



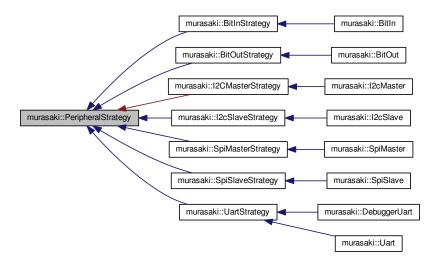
The documentation for this struct was generated from the following file:

• /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/debuggerfifo.hpp

# 12.19 murasaki::PeripheralStrategy Class Reference

#include <peripheralstrategy.hpp>

Inheritance diagram for murasaki::PeripheralStrategy:



## 12.19.1 Detailed Description

This class provides the GetPeripheralHandle() member function as a common stub for the debugging logger. The loggers sometimes refers the raw peripheral to respond to the post mortem situation. By using class, programmer can pass the raw peripheral handler to loggers, while keep it hidden from the application.

The documentation for this class was generated from the following file:

/home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/peripheralstrategy.hpp

## 12.20 murasaki::Platform Struct Reference

#include <platform\_defs.hpp>

Collaboration diagram for murasaki::Platform:



## 12.20.1 Detailed Description

A collection of the peripheral / MPU control variable.

This is a custom struct. Programmer can change this struct as suitable to the hardware and software. But debugger\_ member variable have to be left untouched.

In the run time, the debugger\_variable have to be initialized by appropriate murasaki::Debugger class instance.

## See murasaki::platform

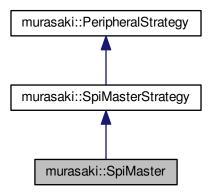
The documentation for this struct was generated from the following file:

/home/takemasa/murasaki samples/nucleo-f746-sample/Inc/platform defs.hpp

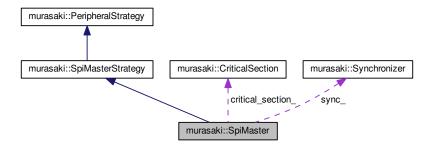
# 12.21 murasaki::SpiMaster Class Reference

#include <spimaster.hpp>

Inheritance diagram for murasaki::SpiMaster:



Collaboration diagram for murasaki::SpiMaster:



#### **Public Member Functions**

- SpiMaster (SPI HandleTypeDef \*spi handle)
- virtual SpiStatus TransmitAndReceive (murasaki::SpiSlaveSpecifierStrategy \*spi\_spec, const uint8\_t \*tx
   \_data, uint8\_t \*rx\_data, unsigned int size, murasaki::WaitMilliSeconds timeout\_ms=murasaki::kwms
   Indefinitely)
- virtual bool TransmitAndReceiveCompleteCallback (void \*ptr)
- virtual bool HandleError (void \*ptr)

## 12.21.1 Detailed Description

The SpiMaster class is the wrapper of the SPI controller. To use the SpiMaster class, make an instance with SPI\_HandleTypeDef \* type pointer. For example, to create an instance for the SPI3 peripheral :

```
my_spi3 = new murasaki::SpiMaster(&hspi3);
```

Where hspi3 is the handle generated by CubeMX for SPI3 peripheral. To use this class, the SPI peripheral have to be configured to use the interrupt and DMA. The bitrate should be configured by the CubeMX.

In addition to the instantiation, we need to prepare an interrupt callback.

```
void HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef * hspi)
{
    my_spi3->TransmitAndReceiveCompleteCallback(hspi);
}
```

Where HAL\_SPI\_TxRxCpltCallback is a predefined name of the SPI interrupt handler. This is invoked by system whenever a interrupt baed SPI transmission is complete. Because the default function is weakly bound, above definition will overwride the default one.

Note that above callback is invoked for any SPIn where n is 1, 2, 3... To avoid the confusion, SpiMaster::Transfer  $\leftarrow$  Complete Callback () method chckes whether given parameter matches with its SPI\_Handle Type Def \* pointer ( which was passed to constructor ). And only when both matches, the member function execute the interrupt termination process.

Once the instance and callbacks are correctly prepared, we can use the Transfer member function.

The SpiMaster::TransmitAndReceive() member function is a blocking function. A programmer can specify the timeout by timeout\_ms parameter. By default, this parameter is set by kwmsIndefinitely which specifes never time out.

Both methods can be called from only the task context. If these are called in the ISR context, the result is unknown.

Note: The behavior of when the timeout happen is not tested. Actually, it should not happen because DMA is taken in SPI transmission. Murasaki stpos internal DMA, interrupt and SPI processing internally then, return.

Other error will cause the re-initializing of the SPI master. Murasaki doesn't support any of CRC detection, TI frame mode or Multi-master SPI.

### 12.21.2 Constructor & Destructor Documentation

```
12.21.2.1 murasaki::SpiMaster::SpiMaster ( SPI_HandleTypeDef * spi_handle )
```

Constractor.

#### **Parameters**

spi handle	Handle to the SPI peripheral.	This have to be configured to use DMA by CubeMX.
- r	-	

### 12.21.3 Member Function Documentation

12.21.3.1 bool murasaki::SpiMaster::HandleError ( void \* ptr ) [virtual]

Error handling.

### **Parameters**

ptr	Pointer to I2C_HandleTypeDef struct.
-----	--------------------------------------

#### Returns

true: ptr matches with device and handle the error. false: doesn't match.

A handle to print out the error message.

Checks whether handle has error and if there is, print appropriate error. Then return.

Implements murasaki::SpiMasterStrategy.

12.21.3.2 SpiStatus murasaki::SpiMaster::TransmitAndReceive ( murasaki::SpiSlaveSpecifierStrategy \* spi\_spec, const uint8\_t \* tx\_data, uint8\_t \* rx\_data, unsigned int size, murasaki::WaitMilliSeconds timeout\_ms = murasaki::kwmsIndefinitely ) [virtual]

Data transfer to/from SPI slave.

## **Parameters**

spi_spec	A pointer to the AbstractSpiSpecification to specify the slave device.
tx_data	Data to be transmitted
rx_data	Data buffer to receive data
size	Transfer data size [byte] for each way.
timeout_ms	Timeout limit [mS]

## Returns

true if transfer complete, false if timeout

Transfer the data to/from SPI slave specified by parameter spi\_spec.

This member funciton re-initialize the SPI peripheral based on the clock information from the spi\_spec. And then, assert the chips elect through the spi\_spec during the data transfer.

Following are the return codes:

- murasaki::kspisOK : The transfer complete without error.
- murasaki::kspisModeCRC : CRC error was detected.
- murasaki::kspisOverflow : SPI overflow or underflow was detected.
- murasaki::kspisFrameError Frame error in TI mode.
- murasaki::kspisDMA: Some DMA error was detected in HAL. SPI re-initialized.
- murasaki::kspisErrorFlag : Unhandled flags. SPI re-initialized.
- murasaki::ki2csTimeOut : Timeout detected. DMA stopped.
- · Other: Unhandled error. SPI re-initialized.

Implements murasaki::SpiMasterStrategy.

12.21.3.3 bool murasaki::SpiMaster::TransmitAndReceiveCompleteCallback(void\*ptr) [virtual]

Callback to notify the end of transfer.

#### **Parameters**

ptr Pointer to the control object.

#### Returns

true if no error.

Implements murasaki::SpiMasterStrategy.

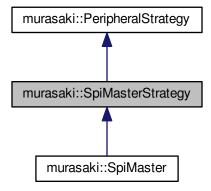
The documentation for this class was generated from the following files:

- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/spimaster.hpp
- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/Src/spimaster.cpp

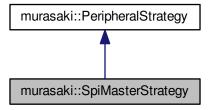
## 12.22 murasaki::SpiMasterStrategy Class Reference

#include <spimasterstrategy.hpp>

Inheritance diagram for murasaki::SpiMasterStrategy:



Collaboration diagram for murasaki::SpiMasterStrategy:



### **Public Member Functions**

- virtual SpiStatus TransmitAndReceive (murasaki::SpiSlaveSpecifierStrategy \*spi\_spec, const uint8\_t \*tx
   \_\_data, uint8\_t \*rx\_data, unsigned int size, murasaki::WaitMilliSeconds timeout\_ms=murasaki::kwms
   Indefinitely)=0
- virtual bool TransmitAndReceiveCompleteCallback (void \*ptr)=0
- virtual bool HandleError (void \*ptr)=0

## 12.22.1 Detailed Description

This class provides a thread safe, blocking SPI transfer.

#### 12.22.2 Member Function Documentation

**12.22.2.1** virtual bool murasaki::SpiMasterStrategy::HandleError ( void \* ptr ) [pure virtual]

Handling error report of device.

#### **Parameters**

#### Returns

true if ptr matches with device and handle the error. false if ptr doesn't match A member function to detect error.

The error handling is depend on the implementation.

Implemented in murasaki::SpiMaster.

Thread safe, blocking SPI transfer.

#### **Parameters**

spi_spec	Pointer to the SPI slave specifier which has clock configuration and chip select handling.
tx_data	Data to be transmitted
rx_data	Data buffer to receive data
size	Transfer data size [byte] for each way. Must be smaller than 65536
timeout_ms	Timeout limit [mS]

#### Returns

true if transfer complete, false if timeout

Implemented in murasaki::SpiMaster.

**12.22.2.3 virtual bool murasaki::SpiMasterStrategy::TransmitAndReceiveCompleteCallback ( void \*** *ptr* **) [pure virtual]** 

Callback to notifiy the end of transfer.

#### **Parameters**

ptr	Pointer to the control object.

#### Returns

true if no error.

Implemented in murasaki::SpiMaster.

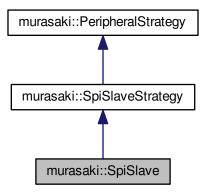
The documentation for this class was generated from the following file:

/home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/spimasterstrategy.hpp

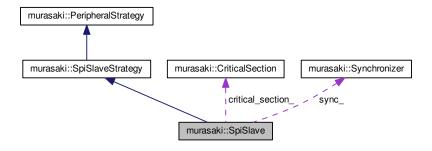
## 12.23 murasaki::SpiSlave Class Reference

#include <spislave.hpp>

Inheritance diagram for murasaki::SpiSlave:



Collaboration diagram for murasaki::SpiSlave:



## **Public Member Functions**

- SpiSlave (SPI\_HandleTypeDef \*spi\_handle)
- virtual SpiStatus TransmitAndReceive (const uint8\_t \*tx\_data, uint8\_t \*rx\_data, unsigned int size, unsigned int \*transfered\_count, murasaki::WaitMilliSeconds timeout\_ms=murasaki::kwmsIndefinitely)
- virtual bool TransmitAndReceiveCompleteCallback (void \*ptr)
- virtual bool HandleError (void \*ptr)

#### 12.23.1 Detailed Description

The SpiSlave class is the wrapper of the SPI controller. To use the SpiSlave class, make an instance with SPI\_← HandleTypeDef \* type pointer. For example, to create an instance for the SPI3 peripheral :

```
my_spi3 = new murasaki::SpiSlave(&hspi3);
```

Where hspi3 is the handle generated by CubeMX for SPI3 peripheral. To use this class, the SPI peripheral have to be configured to use the interrupt and DMA. Also the bitrate, CPOL and CPHA should be configured by the CubeMX.

In addition to the instantiation, we need to prepare an interrupt callback.

```
void HAL_SPI_TxRxCpltCallback (SPI_HandleTypeDef * hspi)
{
    my_spi3->TransmitAndReceiveCompleteCallback(hspi);
}
```

Where HAL\_SPI\_TxRxCpltCallback is a predefined name of the SPI interrupt handler. This is invoked by system whenever a interrupt baed SPI transmission is complete. Because the default function is weakly bound, above definition will override the default one.

Note that above callback is invoked for any SPIn where n is 1, 2, 3... To avoid the confusion, SpiSlave::Transfer ← CompleteCallback() method checkes whether given parameter matches with its SPI\_HandleTypeDef \* pointer ( which was passed to constructor ). And only when both matches, the member function execute the interrupt termination process.

Once the instance and callback are correctly prepared, we can use the Transfer member function.

The SpiSlave::TransmitAndReceive() member function is a blocking function. A programmer can specify the timeout by timeout ms parameter. By default, this parameter is set by kwmsIndefinitely which specifies never time out.

This methods can be called from only the task context. If these are called in the ISR context, the result is unknown.

Other error will cause the re-initializing of the SPI slave. Murasaki doesn't support any of CRC detection, TI frame mode or Multi-master SPI.

## 12.23.2 Constructor & Destructor Documentation

```
12.23.2.1 murasaki::SpiSlave::SpiSlave ( SPI_HandleTypeDef * spi_handle )
```

Constractor.

**Parameters** 

```
spi_handle Handle to the SPI peripheral. This have to be configured to use DMA by CubeMX.
```

### 12.23.3 Member Function Documentation

**12.23.3.1** bool murasaki::SpiSlave::HandleError ( void \* ptr ) [virtual]

Error handling.

#### **Parameters**

ptr	Pointer to I2C_HandleTypeDef struct.
-----	--------------------------------------

#### Returns

true: ptr matches with device and handle the error. false : doesn't match.

A handle to print out the error message.

Checks whether handle has error and if there is, print appropriate error. Then return.

Implements murasaki::SpiSlaveStrategy.

12.23.3.2 SpiStatus murasaki::SpiSlave::TransmitAndReceive ( const uint8\_t \* tx\_data, uint8\_t \* rx\_data, unsigned int size, unsigned int \* transfered\_count, murasaki::WaitMilliSeconds timeout\_ms = murasaki::kwmsIndefinitely )

[virtual]

Data transfer to/from SPI slave.

#### **Parameters**

tx_data	Data to be transmitted
rx_data	Data buffer to receive data
size	Transfer data size [byte] for each way.
transfered_count	( Currently, Just ignored) The transfered number of bytes during API.
timeout_ms	Timeout limit [mS]

## Returns

true if transfer complete, false if timeout

Transfer the data to/from SPI slave specified by parameter spi\_spec.

This member funciton re-initialize the SPI peripheral based on the clock information from the spi\_spec. And then, assert the chips elect through the spi\_spec during the data transfer.

Following are the return codes:

- murasaki::kspisOK : The transfer complete without error.
- murasaki::kspisModeCRC : CRC error was detected.
- murasaki::kspisOverflow : SPI overflow or underflow was detected.
- murasaki::kspisFrameError Frame error in TI mode.
- murasaki::kspisDMA: Some DMA error was detected in HAL. SPI re-initialized.

- murasaki::kspisErrorFlag : Unhandled flags. SPI re-initialized.
- murasaki::ki2csTimeOut : Timeout detected. DMA stopped.
- · Other: Unhandled error. SPI re-initialized.

Implements murasaki::SpiSlaveStrategy.

12.23.3.3 bool murasaki::SpiSlave::TransmitAndReceiveCompleteCallback (void \* ptr ) [virtual]

Callback to notify the end of transfer.

#### **Parameters**

ptr Pointer to the control object.

#### Returns

true if no error.

Implements murasaki::SpiSlaveStrategy.

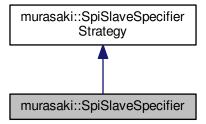
The documentation for this class was generated from the following files:

- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/Inc/spislave.hpp
- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/Src/spislave.cpp

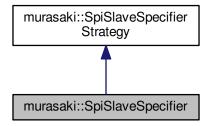
## 12.24 murasaki::SpiSlaveSpecifier Class Reference

#include <spislavespecifier.hpp>

Inheritance diagram for murasaki::SpiSlaveSpecifier:



Collaboration diagram for murasaki::SpiSlaveSpecifier:



#### **Public Member Functions**

- SpiSlaveSpecifier (murasaki::SpiClockPolarity pol, murasaki::SpiClockPhase pha,::GPIO\_TypeDef \*port, uint16 t pin)
- SpiSlaveSpecifier (unsigned int pol, unsigned int pha,::GPIO\_TypeDef \*const port, uint16\_t pin)
- virtual void AssertCs ()
- virtual void DeassertCs ()

## 12.24.1 Detailed Description

This class describes how the slave is. The description is clock POL and PHA for the speicific slave device.

In addition to the clock porality, the instans of this class works as salogate of the chip select control.

The instans will be passed to the SpiMaster class.

## 12.24.2 Constructor & Destructor Documentation

12.24.2.1 murasaki::SpiSlaveSpecifier::SpiSlaveSpecifier ( murasaki::SpiClockPolarity pol, murasaki::SpiClockPhase pha, ::GPIO\_TypeDef \* port, uint16\_t pin )

#### Constructor.

## **Parameters**

pol	Polarity setting
pha	Phase setting
port	GPIO port of the chip select
pin	GPIO pin of the chip select

The port and pin parameters are passed to the HAL\_GPIO\_WritePin(). The port and pin have to be configured by CubeMX correctly.

12.24.2.2 murasaki::SpiSlaveSpecifier::SpiSlaveSpecifier ( unsigned int *pol*, unsigned int *pha*, ::GPIO\_TypeDef \*const *port*, uint16\_t *pin* )

#### Constructor.

#### **Parameters**

pol	Polarity setting
pha	Phase setting
port	GPIO port of the chip select
pin	GPIO pin of the chip select

The port and pin parameters are passed to the HAL\_GPIO\_WritePin(). The port and pin have to be configured by CubeMX correctly.

#### 12.24.3 Member Function Documentation

12.24.3.1 void murasaki::SpiSlaveSpecifier::AssertCs() [virtual]

Chip select assertion.

This member function asset the output line to select the slave chip.

Reimplemented from murasaki::SpiSlaveSpecifierStrategy.

12.24.3.2 void murasaki::SpiSlaveSpecifier::DeassertCs() [virtual]

Chip select deassertoin.

This member function deasset the output line to de-select the slave chip.

Reimplemented from murasaki::SpiSlaveSpecifierStrategy.

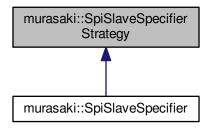
The documentation for this class was generated from the following files:

- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/spislavespecifier.hpp
- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/Src/spislavespecifier.cpp

## 12.25 murasaki::SpiSlaveSpecifierStrategy Class Reference

#include <spislavespecifierstrategy.hpp>

Inheritance diagram for murasaki::SpiSlaveSpecifierStrategy:



#### **Public Member Functions**

- SpiSlaveSpecifierStrategy (murasaki::SpiClockPolarity pol, murasaki::SpiClockPhase pha)
- SpiSlaveSpecifierStrategy (unsigned int pol, unsigned int pha)
- virtual void AssertCs ()
- virtual void DeassertCs ()
- murasaki::SpiClockPhase GetCpha ()
- murasaki::SpiClockPolarity GetCpol ()

## 12.25.1 Detailed Description

A prototype of the SPI slave device specifier.

The specifier adds the following SPI attributes :

- CPOL
- CPHA
- · Chip select control for slave.

Because SPI slave has different setting device by device, this specifier should be passed to the each transactions.

AssetCs() and DeassertCs() have to be overriden to control the chip select output. These member functions will be called from the AbstractSpiMaster.

#### 12.25.2 Constructor & Destructor Documentation

12.25.2.1 murasaki::SpiSlaveSpecifierStrategy::SpiSlaveSpecifierStrategy ( murasaki::SpiClockPolarity pol, murasaki::SpiClockPhase pha )

Constructor.

#### **Parameters**

pol	Polarity setting
pha	Phase setting

12.25.2.2 murasaki::SpiSlaveSpecifierStrategy::SpiSlaveSpecifierStrategy ( unsigned int pol, unsigned int pha )

Constructor.

#### **Parameters**

pol	Polarity setting
pha	Phase setting

#### 12.25.3 Member Function Documentation

12.25.3.1 void murasaki::SpiSlaveSpecifierStrategy::AssertCs() [virtual]

Chip select assertion.

This member function asset the output line to select the slave chip.

This have to be overriden.

Reimplemented in murasaki::SpiSlaveSpecifier.

12.25.3.2 void murasaki::SpiSlaveSpecifierStrategy::DeassertCs() [virtual]

Chip select deassertoin.

This member function deasset the output line to de-select the slave chip.

This have to be overriden.

Reimplemented in murasaki::SpiSlaveSpecifier.

12.25.3.3 murasaki::SpiClockPhase murasaki::SpiSlaveSpecifierStrategy::GetCpha ( )

Getter of the CPHA.

Returns

**CPHA** setting

12.25.3.4 murasaki::SpiClockPolarity murasaki::SpiSlaveSpecifierStrategy::GetCpol()

Getter of the CPOL.

Returns

**CPOL** setting

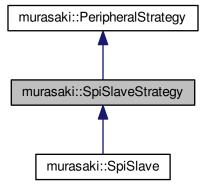
The documentation for this class was generated from the following files:

- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/spislavespecifierstrategy.hpp
- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/Src/spislavespecifierstrategy.cpp

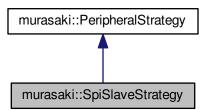
## 12.26 murasaki::SpiSlaveStrategy Class Reference

#include <spislavestrategy.hpp>

Inheritance diagram for murasaki::SpiSlaveStrategy:



Collaboration diagram for murasaki::SpiSlaveStrategy:



#### **Public Member Functions**

- virtual SpiStatus TransmitAndReceive (const uint8\_t \*tx\_data, uint8\_t \*rx\_data, unsigned int size, unsigned int \*transfered\_count=nullptr, murasaki::WaitMilliSeconds timeout\_ms=murasaki::kwmsIndefinitely)=0
- virtual bool TransmitAndReceiveCompleteCallback (void \*ptr)=0
- virtual bool HandleError (void \*ptr)=0

### 12.26.1 Detailed Description

This class provides a thread safe, blocking SPI transfer.

#### 12.26.2 Member Function Documentation

**12.26.2.1** virtual bool murasaki::SpiSlaveStrategy::HandleError ( void \* ptr ) [pure virtual]

Handling error report of device.

#### **Parameters**

Pointer for generic use. Usually, points a struct of a device control
---

#### Returns

true if ptr matches with device and handle the error. false if ptr doesn't match A member function to detect error.

The error handling is depend on the implementation.

Implemented in murasaki::SpiSlave.

12.26.2.2 virtual SpiStatus murasaki::SpiSlaveStrategy::TransmitAndReceive ( const uint8\_t \* tx\_data, uint8\_t \* rx\_data, unsigned int size, unsigned int \* transfered\_count = nullptr, murasaki::WaitMilliSeconds timeout\_ms = murasaki::kwmsIndefinitely ) [pure virtual]

Thread safe, blocking SPI transfer.

#### **Parameters**

tx_data	Data to be transmitted
rx_data	Data buffer to receive data
size	Transfer data size [byte] for each way. Must be smaller than 65536
transfered_count	The transfered number of bytes during API.
timeout_ms	Timeout limit [mS]

#### Returns

true if transfer complete, false if timeout

Implemented in murasaki::SpiSlave.

**12.26.2.3** virtual bool murasaki::SpiSlaveStrategy::TransmitAndReceiveCompleteCallback (void \* ptr ) [pure virtual]

Callback to notifiy the end of transfer.

#### **Parameters**

#### Returns

true if no error.

Implemented in murasaki::SpiSlave.

The documentation for this class was generated from the following file:

/home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/spislavestrategy.hpp

## 12.27 murasaki::Synchronizer Class Reference

```
#include <synchronizer.hpp>
```

#### **Public Member Functions**

- bool Wait (WaitMilliSeconds timeout\_ms=kwmsIndefinitely)
- void Release ()

## 12.27.1 Detailed Description

Synchronization mean, task waits for a interrupt by calling InterruptSynchronizer::WaitForInterruptFromTask() and during the wait, task yields the cpu to other task. So, CPU can do other job during a task is waiting for interrupt. Interrupt will allow task run again by InterruptSynchronizer::ReleasetaskFromISR() member function.

## 12.27.2 Member Function Documentation

```
12.27.2.1 void murasaki::Synchronizer::Release ( )
```

Release the task.

Release the task waiting. This member function must be called from both task and the interrupt context.

12.27.2.2 bool murasaki::Synchronizer::Wait ( WaitMilliSeconds timeout\_ms = kwmsIndefinitely )

Let the task wait for an interrupt.

#### **Parameters**

timeout_ms   Timeout by millisecond. The default value let the task wait for	for interrupt forever.
--	------------------------

#### Returns

True if interrupt came before timeout. False if timeout happen.

This member function have to be called from the task context. Otherwise, the behavior is not predictable.

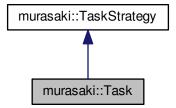
The documentation for this class was generated from the following files:

- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/synchronizer.hpp
- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/Src/synchronizer.cpp

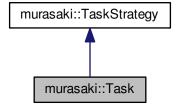
## 12.28 murasaki::Task Class Reference

#include <task.hpp>

Inheritance diagram for murasaki::Task:



Collaboration diagram for murasaki::Task:



#### **Public Member Functions**

Task (const char \*task\_name, unsigned short stack\_depth, UBaseType\_t task\_priority, const void \*task\_
parameter, void(\*task\_body\_func)(const void \*))

#### **Protected Member Functions**

• virtual void TaskBody (const void \*ptr)

#### **Additional Inherited Members**

## 12.28.1 Detailed Description

This is handy class to encapsulate the task creation without inheriting. A task can be created easy like:

Then, task you can call Start() member function to run.

```
murasaki::platform.task1->Start();
```

## 12.28.2 Constructor & Destructor Documentation

12.28.2.1 murasaki::Task::Task ( const char \* task\_name, unsigned short stack\_depth, UBaseType\_t task\_priority, const void \* task\_parameter, void(\*)(const void \*) task\_body\_func )

Ease to use task class.

#### **Parameters**

task_name	A name of task. This is relevant to the FreeRTOS's API manner.
stack_depth	Task stack size by byte.
task_priority	The task priority. Max priority is defined by configMAX_PRIOIRTIES in FreeRTOSConfig.h
task_parameter	A pointer to the parameter passed to task.
task_body_func	A pointer to the task body function.

Create an task object. Given parameters are stored internally. And then passed to the FreeRTOS API when task is started by Start() member function.

A task parameter can be passed to task through the task\_parameter. This pointer is simply passed to the task body function without modification.

#### 12.28.3 Member Function Documentation

12.28.3.1 void murasaki::TaskBody ( const void \* ptr ) [protected], [virtual]

Task member function.

#### **Parameters**

ptr The task\_parameter parameter of the constructor is passed to this parameter.

This member function runs as task. In this function, the function passed thorough task\_body\_func parameter is invoked as actual task body.

Implements murasaki::TaskStrategy.

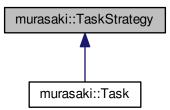
The documentation for this class was generated from the following files:

- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/task.hpp
- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/Src/task.cpp

## 12.29 murasaki::TaskStrategy Class Reference

#include <taskstrategy.hpp>

Inheritance diagram for murasaki::TaskStrategy:



#### **Public Member Functions**

- TaskStrategy (const char \*task\_name, unsigned short stack\_depth, UBaseType\_t task\_priority, const void \*task\_parameter)
- void Start ()
- const char \* GetName ()

#### **Protected Member Functions**

• virtual void TaskBody (const void \*ptr)=0

#### **Static Protected Member Functions**

static void Launch (void \*ptr)

#### 12.29.1 Detailed Description

Encapsulate a FreeRTOS task.

The constructor just stores given parameter internally. And then, these parameter is passed to a task when Start() member function is called. Actual task creation is done inside Start().

The destructor deletes the task. Releasing thask from all the resources (ex: semaphore) before deleting, is the responsibility of the programmer.

Base on the description at http://idken.net/posts/2017-02-01-freertos\_task\_cpp/

#### 12.29.2 Constructor & Destructor Documentation

12.29.2.1 murasaki::TaskStrategy::TaskStrategy ( const char \* task\_name, unsigned short stack\_depth, UBaseType\_t task\_priority, const void \* task\_parameter )

Constractor. Task entity is not created here.

#### **Parameters**

task_name	Name of task. Will be passed to task when started.
stack_depth	[Byte]
task_priority	Priority of the task. from 1 to up to configMAX_PRIORITIES -1. The high number is the high priority.
task_parameter	Optional parameter to the task.

## 12.29.3 Member Function Documentation

12.29.3.1 const char \* murasaki::TaskStrategy::GetName( )

Get a name of task.

#### Returns

A name of task.

12.29.3.2 void murasaki::TaskStrategy::Launch(void\*ptr) [static], [protected]

Internal use only. Create a task from TaskBody()

#### **Parameters**

ptr	passing "this" pointer.
-----	-------------------------

12.29.3.3 void murasaki::TaskStrategy::Start ( )

Create a task and run it.

A task is created with given parameter to the constructors and then run.

12.29.3.4 virtual void murasaki::TaskStrategy::TaskBody ( const void \* ptr ) [protected], [pure virtual]

Actual task entity. Must be overridden by programmer.

#### **Parameters**

ptr Optional parameter to the task body. This ptr is copied from the task\_parameter of the Constructor.

The task body is called only once as task entity. Programmer have to override this member function with his/her own TaskBody().

From this member function, class members are able to access.

Implemented in murasaki::Task.

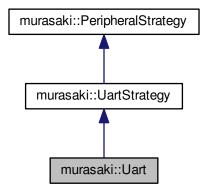
The documentation for this class was generated from the following files:

- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/taskstrategy.hpp
- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/Src/taskstrategy.cpp

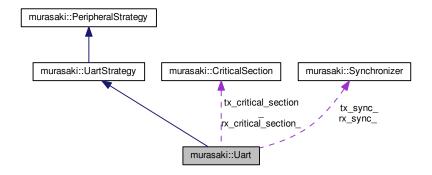
#### 12.30 murasaki::Uart Class Reference

#include <uart.hpp>

Inheritance diagram for murasaki::Uart:



#### Collaboration diagram for murasaki::Uart:



## **Public Member Functions**

- Uart (UART\_HandleTypeDef \*uart)
- virtual void SetHardwareFlowControl (UartHardwareFlowControl control)
- virtual void SetSpeed (unsigned int baud\_rate)
- virtual murasaki::UartStatus Transmit (const uint8\_t \*data, unsigned int size, WaitMilliSeconds timeout\_ms)
- virtual murasaki::UartStatus Receive (uint8\_t \*data, unsigned int count, unsigned int \*transfered\_count, UartTimeout uart\_timeout, WaitMilliSeconds timeout\_ms)
- virtual bool TransmitCompleteCallback (void \*const ptr)
- virtual bool ReceiveCompleteCallback (void \*const ptr)
- virtual bool HandleError (void \*const ptr)

#### 12.30.1 Detailed Description

The Uart class is the wrapper of the UART controller. To use the Uart class, make an instance with UART\_Handle 

TypeDef ∗ type pointer. For example, to create an instance for the UART3 peripheral :

```
my_uart3 = new murasaki::Uart(&huart3);
```

Where huart3 is the handle generated by CubeMX for UART3 peripheral. To use this class, the UART peripheral have to be configured to use the DMA functionality. The baud rate, length and flow control should be configured by the CubeMX.

In addition to the instantiation, we need to prepare an interrupt callback.

```
void HAL_UART_TxCpltCallback(UART_HandleTypeDef * huart)
{
    my_uart3->TransmitCompleteCallback(huart);
}
```

Where HAL\_UART\_TxCpltCallback is a predefined name of the UART interrupt handler. This is invoked by system whenever a DMA baed UART transmission is complete. Becuase the default function is weakly bound, above definition will overwride the default one.

Note that above callback is invoked for any UARTn where n is 1, 2, 3... To avoid the confusion, Uart::Transmit← CompleteCallback() method chckes whether given parameter matches with its UART\_HandleTypeDef \* pointer ( which was passed to constructor ). And only when both matches, the member function execute the interrupt termination process.

As same as Tx, RX needs HAL UART TxCpltCallback().

Once the instance and callbacks are correctly prepared, we can use the Tx/Rx member function.

The Uart::Transmit() member function is a blocking function. A programmer can specify the timeout by timeout\_ms parameter. By default, this parameter is set by kwmsIndefinitely which specifes never time out.

The Uart::Receive() member function is a blocking function. A programmer can specify the timeout by timeout\_ms parameter. By default, this parameter is set by kwmsIndefinitely which specifes never time out.

Both methods can be called from only the task context. If these are called in the ISR context, the result is unknown.

#### 12.30.2 Constructor & Destructor Documentation

```
12.30.2.1 murasaki::Uart::Uart ( UART_HandleTypeDef * uart )
```

Constructor.

#### **Parameters**

uart	Pointer to a UART control struct. This device have to be configured to use DMA and interrupt for both Tx
	and Rx.

Store the given uart pointer into the internal variable. This pointer is passed to the STM32Cube HAL UART functions when needed.

#### 12.30.3 Member Function Documentation

12.30.3.1 bool murasaki::Uart::HandleError ( void \*const ptr ) [virtual]

Error handling.

#### **Parameters**

ptr	Pointer to UART_HandleTypeDef struct.
-----	---------------------------------------

#### Returns

true: ptr matches with UART device and handle the error. false : doesn't match.

A handle to print out the error message.

Checks whether handle has error and if there is, print appropriate error. Then return.

Implements murasaki::UartStrategy.

12.30.3.2 murasaki::UartStatus murasaki::Uart::Receive ( uint8\_t \* data, unsigned int count, unsigned int \* transfered\_count, UartTimeout uart\_timeout, WaitMilliSeconds timeout\_ms ) [virtual]

Receive raw data through an UART by blocking mode.

#### **Parameters**

data	Data buffer to place the received data
count	The count of the data (byte) to be transfered. Must be smaller than 65536
transfered_count	( Currently, Just ignored) Number of bytes transfered. The nullPtr means no need to return
	value.
uart_timeout	Specify murasaki::kutldleTimeout, if idle line timeout is needed.
timeout_ms	Time out limit by milliseconds.

#### Returns

True if all data transfered completely. False if time out happen.

Receive to given data buffer through an UART device.

The receiving mode is blocking. That means, function returns when specified number of data has been received, except timeout. Passing murasaki::kwmsIndefinitely to the parameter timeout\_ms orders not to return until complete receiving. Other value of timeout\_ms parameter specifies the time out by millisecond. If time out happen, function returns false. If not happen, it returns true.

This function is exclusive. Internally this function is guarded by mutex. Then this function is thread safe. This function is forbidden to call from ISR.

The retun values are:

- murasaki::kursOK : Transmit complete.
- murasaki::kursTimeOut : Time out occur.
- murasaki::kursOverrun: Next char was written to TX register. This is fatal problem in HAL. Periperal is re-initialized internally.
- murasaki::kursDMA: This is fatal problem in HAL. Peripheral is re-initialized internally.
- other: This is fatal problem in HAL. Peripheral is re-initialized internally.

Implements murasaki::UartStrategy.

12.30.3.3 bool murasaki::Uart::ReceiveCompleteCallback (void \*const ptr ) [virtual]

Call back for entire block transfer completion.

#### **Parameters**

```
ptr Pointer to UART_HandleTypeDef struct.
```

#### Returns

true: ptr matches with UART device and handle the call back. false : doesn't match.

A call back to notify the end of entire block transfer. This is considered as the end of DMA based receiving. The context have to be interrupt.

This member function checks whether the given ptr parameter matches its own device, and if matched, Release the waiting task and return true. If it doesn't match, just return false.

This method have to be called from HAL\_UART\_RxCpltCallback(). See STM32F7 HAL manual for detail

Implements murasaki::UartStrategy.

12.30.3.4 void murasaki::Uart::SetHardwareFlowControl ( UartHardwareFlowControl control ) [virtual]

Set the behavior of the hardware flow control.

#### **Parameters**

control The control mode.

Before calling this method, all transmission and recevie activites have to be finished. This is responsibility of the programmer.

Note this method is NOT re-etnrant. In other word, this member function can be called from both task and interrupt context.

Reimplemented from murasaki::UartStrategy.

12.30.3.5 void murasaki::Uart::SetSpeed ( unsigned int baud\_rate ) [virtual]

Set the BAUD rate.

#### **Parameters**

```
baud_rate | BAUD rate ( 110, 300,... 57600,... )
```

Before calling this method, all transmission and recevie activites have to be finished. This is responsibility of the programmer.

Note this method is NOT re-ethrant. In other word, this member function can be called from both task and interrupt context.

Reimplemented from murasaki::UartStrategy.

12.30.3.6 murasaki::UartStatus murasaki::Uart::Transmit ( const uint8\_t \* data, unsigned int size, WaitMilliSeconds timeout\_ms ) [virtual]

Transmit raw data through an UART by blocking mode.

#### **Parameters**

data	Data buffer to be transmitted.
size	The count of the data (byte) to be transfered. Must be smaller than 65536
timeout_ms	Time out limit by milliseconds.

#### Returns

True if all data transfered completely. False if time out happen.

Transmit given data buffer through an UART device.

The transmission mode is blocking. That means, function returns when all data has been transmitted, except timeout. Passing murasaki::kwmsIndefinitely to the parameter timeout\_ms orders not to return until complete transmission. Other value of timeout\_ms parameter specifies the time out by millisecond. If time out happen, function returns false. If not happen, it returns true.

This function is exclusive. Internally the function is guarded by mutex. Then this function is thread safe. This function is forbiddedn to call from ISR.

Implements murasaki::UartStrategy.

12.30.3.7 bool murasaki::Uart::TransmitCompleteCallback (void \*const ptr) [virtual]

Call back for entire block transfer completion.

#### **Parameters**

tr Pointer to UART_HandleTypeDef struct.	
--	--

#### Returns

true: ptr matches with UART device and handle the call back. false : doesn't match.

A call back to notify the end of entire block transfer. This is considered as the end of DMA based transmission. The context have to be interrupt.

This member function checks whether the given ptr parameter matches its own device, and if matched, Release the waiting task and return true. If it doesn't match, just return false.

This method have to be called from HAL\_UART\_TxCpltCallback(). See STM32F7 HAL manual for detail

The retun values are:

- murasaki::kursOK : Received complete.
- murasaki::kursTimeOut : Time out occur.
- murasaki::kursFrame : Receive error by wrong word size configuration.
- · murasaki::kursParity : Parity error.
- murasaki::kursNoise : Error by noise.
- murasaki::kursDMA: This is fatal problem in HAL. Peripheral is re-initialized internally.
- other: This is fatal problem in HAL. Peripheral is re-initialized internally.

Implements murasaki::UartStrategy.

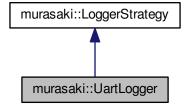
The documentation for this class was generated from the following files:

- /home/takemasa/murasaki samples/nucleo-f746-sample/murasaki/lnc/uart.hpp
- /home/takemasa/murasaki samples/nucleo-f746-sample/murasaki/Src/uart.cpp

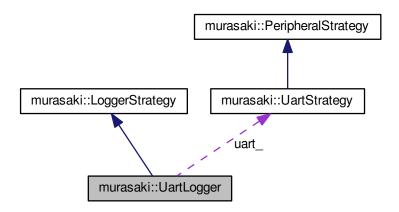
## 12.31 murasaki::UartLogger Class Reference

#include <uartlogger.hpp>

Inheritance diagram for murasaki::UartLogger:



Collaboration diagram for murasaki::UartLogger:



#### **Public Member Functions**

- UartLogger (UartStrategy \*uart)
- virtual void putMessage (char message[], unsigned int size)
- virtual char getCharacter ()
- virtual void DoPostMortem (void \*debugger\_fifo)

## 12.31.1 Detailed Description

This is a standard logging class through the UART port. The instance of this class can be passed to the murasaki← ::Debugger constructor.

See Application Specific Platform as usage example.

#### 12.31.2 Constructor & Destructor Documentation

12.31.2.1 murasaki::UartLogger::UartLogger ( UartStrategy \* uart )

Constructor.

**Parameters** 

uart	Pointer to the uart object.

## 12.31.3 Member Function Documentation

12.31.3.1 void murasaki::UartLogger::DoPostMortem ( void \* debugger\_fifo ) [virtual]

Start post mortem process.

#### **Parameters**

debugger_fifo	Pointer to the DebuggerFifo class object. The data inside this FIFO will be sent to UART This	1
	member function read the data in given FIFO, and then do the auto history.	

This funciton call the DebuggerFifo::SetPostMortem() intenally. Then, output the data inside FIFO through the given UART.

Once all the data is output, this function wait for a receive data. Once data received, this function rewind the FIFO and then, start to transmit the data again.

Reimplemented from murasaki::LoggerStrategy.

12.31.3.2 char murasaki::UartLogger::getCharacter() [virtual]

Character input member function.

#### Returns

A character from input is returned.

This function is considered as blocking. That mean, the function will wait for any user input forever.

Implements murasaki::LoggerStrategy.

12.31.3.3 void murasaki::UartLogger::putMessage ( char message[], unsigned int size ) [virtual]

Message output member function.

#### **Parameters**

message	Non null terminated character array. This data is stored or output to the logger.
size	Size of the message[bytes]. Must be smaller than 65536

Implements murasaki::LoggerStrategy.

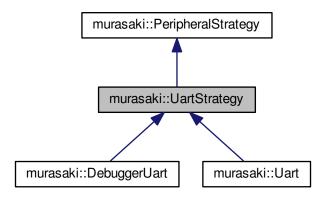
The documentation for this class was generated from the following files:

- /home/takemasa/murasaki samples/nucleo-f746-sample/murasaki/Inc/uartlogger.hpp
- /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/Src/uartlogger.cpp

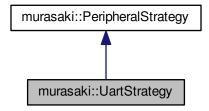
## 12.32 murasaki::UartStrategy Class Reference

#include <uartstrategy.hpp>

Inheritance diagram for murasaki::UartStrategy:



Collaboration diagram for murasaki::UartStrategy:



#### **Public Member Functions**

- virtual void SetHardwareFlowControl (UartHardwareFlowControl control)
- virtual void SetSpeed (unsigned int speed)
- virtual murasaki::UartStatus Transmit (const uint8\_t \*data, unsigned int size, WaitMilliSeconds timeout\_

   ms=murasaki::kwmsIndefinitely)=0
- virtual murasaki::UartStatus Receive (uint8\_t \*data, unsigned int size, unsigned int \*transfered\_count=nullptr, UartTimeout uart\_timeout=murasaki::kutNoldleTimeout, WaitMilliSeconds timeout\_ms=murasaki::kwms← Indefinitely)=0
- virtual bool TransmitCompleteCallback (void \*ptr)=0
- virtual bool ReceiveCompleteCallback (void \*ptr)=0
- virtual bool HandleError (void \*ptr)=0

#### 12.32.1 Detailed Description

A prototype of the UART device. The abstract class shows the usage of the UART peripheral.

This prototype assumes the derived class will transmit / receive data in the task context on RTOS. And both method should be blocking. That men, until the transmit / receve terminates, both method doesn't return.

Two call back methods are prepared to sync with the interrutp which tells the end of Transmit/Recieve.

#### 12.32.2 Member Function Documentation

12.32.2.1 virtual bool murasaki::UartStrategy::HandleError(void\*ptr) [pure virtual]

Handling error report of device.

#### **Parameters**

#### Returns

true if ptr matches with device and handle the error. false if ptr doesn't match A member function to detect error.

The error handling is depend on the implementation.

Implemented in murasaki::Uart, and murasaki::DebuggerUart.

12.32.2.2 virtual murasaki::UartStatus murasaki::UartStrategy::Receive ( uint8\_t \* data, unsigned int size, unsigned int \* transfered\_count = nullptr, UartTimeout uart\_timeout = murasaki::kutNoldleTimeout,

WaitMilliSeconds timeout\_ms = murasaki::kwmsIndefinitely ) [pure virtual]

buffer receive over the UART. Blocking

#### **Parameters**

data	Pointer to the buffer to save the received data.	
size	Number of the data to be received.	
transfered_count	Number of bytes transfered. The nullPtr means no need to return value.	
uart_timeout	Specify murasaki::kutldleTimeout, if idle line timeout is needed.	
timeout_ms	Time out by milli Second.	

#### Returns

Status of the IO processing

Implemented in murasaki::Uart, and murasaki::DebuggerUart.

```
12.32.2.3 virtual bool murasaki::UartStrategy::ReceiveCompleteCallback(void*ptr) [pure virtual]
```

Call back to be called for entire block transfer is complete.

#### **Parameters**

```
ptr Pointer for generic use. Usually, points a struct of a UART device control
```

#### Returns

true: ptr matches with UART device and handle the call back. false: doesn't match.

A call back to notify the end of entire block or byte transfer. The definition of calling timing is depend on the implementation. This is called from an DMA ISR.

Typically, an implementation may check whether the given ptr parameter matches its own device, and if matched, handle it and return true. If it doesn't match, just return false.

Implemented in murasaki::Uart, and murasaki::DebuggerUart.

```
12.32.2.4 virtual void murasaki::UartStrategy::SetHardwareFlowControl ( UartHardwareFlowControl control ) [inline],[virtual]
```

Set the behavior of the hardware flow control.

#### **Parameters**

```
control The control mode.
```

Reimplemented in murasaki::DebuggerUart, and murasaki::Uart.

```
12.32.2.5 virtual void murasaki::UartStrategy::SetSpeed (unsigned int speed) [inline], [virtual]
```

the baud rate

#### **Parameters**

```
speed BAUD rate ( 110, 300, ... 9600,... )
```

Reimplemented in murasaki::DebuggerUart, and murasaki::Uart.

```
12.32.2.6 virtual murasaki::UartStatus murasaki::UartStrategy::Transmit ( const uint8_t * data, unsigned int size, WaitMilliSeconds timeout_ms = murasaki::kwmsIndefinitely ) [pure virtual]
```

buffer transmission over the UART. Blocking

#### **Parameters**

data	Pointer to the buffer to be sent.
size	Number of the data to be sent.
timeout_ms	Time out by mili Second.

#### Returns

Status of the IO processing

Implemented in murasaki::DebuggerUart, and murasaki::Uart.

12.32.2.7 virtual bool murasaki::UartStrategy::TransmitCompleteCallback ( void \* ptr ) [pure virtual]

Call back to be called notify the transfer is complete.

#### **Parameters**

ptr	Pointer for generic use. Usually, points a struct of a UART device control
-----	--

#### Returns

true: ptr matches with UART device and handle the call back. false : doesn't match.

A call back to notify the end of entire block or byte transfer. The definition of calling timing is depend on the implementation. This is called from an DMA ISR.

Typically, an implementation may check whether the given ptr parameter matches its own device, and if matched, handle it and return true. If it doesn't match, just return false.

Implemented in murasaki::Uart, and murasaki::DebuggerUart.

The documentation for this class was generated from the following file:

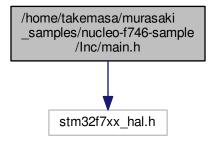
/home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/uartstrategy.hpp

## **Chapter 13**

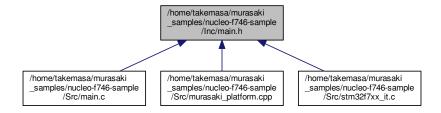
## **File Documentation**

## 13.1 /home/takemasa/murasaki\_samples/nucleo-f746-sample/lnc/main.h File Reference

#include "stm32f7xx\_hal.h"
Include dependency graph for main.h:



This graph shows which files directly or indirectly include this file:



#### **Functions**

• void Error\_Handler (void)

156 File Documentation

#### 13.1.1 Detailed Description

This notice applies to any and all portions of this file that are not between comment pairs USER CODE BEGIN and USER CODE END. Other portions of this file, whether inserted by the user or by software development tools are owned by their respective copyright owners.

Copyright (c) 2019 STMicroelectronics International N.V. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted, provided that the following conditions are met:

- Redistribution of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3. Neither the name of STMicroelectronics nor the names of other contributors to this software may be used to endorse or promote products derived from this software without specific written permission.
- 4. This software, including modifications and/or derivative works of this software, must execute solely and exclusively on microcontroller or microprocessor devices manufactured by or for STMicroelectronics.
- 5. Redistribution and use of this software other than as permitted under this license is void and will automatically terminate your rights under this license.

THIS SOFTWARE IS PROVIDED BY STMICROELECTRONICS AND CONTRIBUTORS "AS IS" AND ANY E  $\leftarrow$  XPRESS, IMPLIED OR STATUTORY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED W  $\leftarrow$  ARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS ARE DISCLAIMED TO THE FULLEST EXTENT PER  $\leftarrow$  MITTED BY LAW. IN NO EVENT SHALL STMICROELECTRONICS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, B  $\leftarrow$  UT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WH  $\leftarrow$  ETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

#### 13.1.2 Function Documentation

13.1.2.1 void Error\_Handler (void)

This function is executed in case of error occurrence.

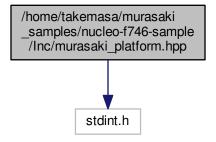
Return values

None

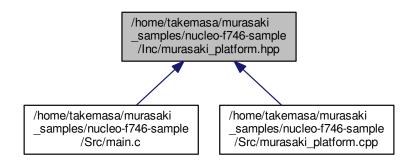
# 13.2 /home/takemasa/murasaki\_samples/nucleo-f746-sample/Inc/murasaki\_platform.hpp File Reference

#include <stdint.h>

Include dependency graph for murasaki\_platform.hpp:



This graph shows which files directly or indirectly include this file:



## **Functions**

- void InitPlatform ()
- void ExecPlatform ()
- void CustomAssertFailed (uint8\_t \*file, uint32\_t line)
- void CustomDefaultHandler ()

## 13.2.1 Detailed Description

158 File Documentation

Date

2017/11/12

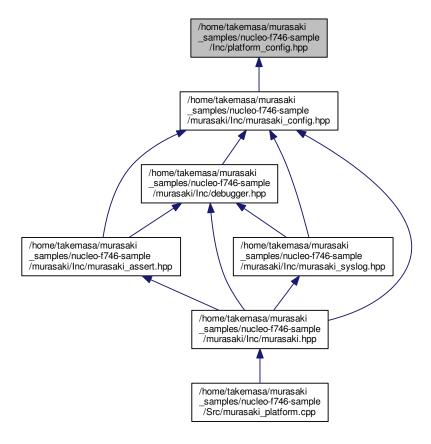
Author

takemasa

The resources below are impremented in the murasaki\_platform.cpp and serve as glue to the main.c.

# 13.3 /home/takemasa/murasaki\_samples/nucleo-f746-sample/Inc/platform\_config.hpp File Reference

This graph shows which files directly or indirectly include this file:



## **Macros**

• #define MURASAKI\_CONFIG\_NOSYSLOG false

## 13.3.1 Detailed Description

Date

2018/01/07

**Author** 

takemasa

If you want to override the macro definition inside platform\_config.hpp, add your definition here.

#### 13.3.2 Macro Definition Documentation

#### 13.3.2.1 #define MURASAKI\_CONFIG\_NOSYSLOG false

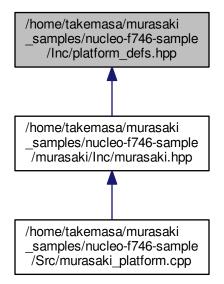
Surpress MURASAKI\_SYSLOG macro.

Set this macro to true, to discard the MURASAKI\_SYSLOG. Set this macro false, to use the syslog.

To override the definition here, define same macro inside platform config.hpp.

# 13.4 /home/takemasa/murasaki\_samples/nucleo-f746-sample/lnc/platform\_defs.hpp File Reference

This graph shows which files directly or indirectly include this file:



#### Classes

· struct murasaki::Platform

#### **Namespaces**

· murasaki

#### **Variables**

• Platform murasaki::platform

## 13.4.1 Detailed Description

Date

2018/01/16

#### Author

takemasa

This file contains user defined struct murasaki::Platform.

This file will be included by murasaki.hpp.

## 13.5 /home/takemasa/murasaki\_samples/nucleo-f746-sample/lnc/stm32f7xx\_hal\_conf.h File Reference

```
#include "stm32f7xx_hal_rcc.h"
#include "stm32f7xx_hal_gpio.h"
#include "stm32f7xx_hal_dma.h"
#include "stm32f7xx_hal_cortex.h"
#include "stm32f7xx_hal_flash.h"
#include "stm32f7xx_hal_i2c.h"
#include "stm32f7xx_hal_pwr.h"
#include "stm32f7xx_hal_spi.h"
#include "stm32f7xx_hal_tim.h"
#include "stm32f7xx_hal_tim.h"
#include dependency graph for stm32f7xx hal conf.h:
```



#### **Macros**

- #define HSE\_VALUE ((uint32\_t)8000000U)
- #define HSE\_STARTUP\_TIMEOUT ((uint32\_t)100U)
- #define HSI VALUE ((uint32 t)16000000U)
- #define LSI\_VALUE ((uint32\_t)32000U)
- #define LSE VALUE ((uint32 t)32768U)
- #define LSE STARTUP TIMEOUT ((uint32 t)5000U)
- #define EXTERNAL\_CLOCK\_VALUE ((uint32\_t)12288000U)
- #define VDD\_VALUE ((uint32\_t)3300U)
- #define TICK\_INT\_PRIORITY ((uint32\_t)0U)
- #define PHY BCR ((uint16 t)0x0000U)
- #define PHY BSR ((uint16 t)0x0001U)
- #define PHY\_RESET ((uint16\_t)0x8000U)
- #define PHY\_LOOPBACK ((uint16\_t)0x4000U)
- #define PHY FULLDUPLEX 100M ((uint16 t)0x2100U)
- #define PHY\_HALFDUPLEX\_100M ((uint16\_t)0x2000U)
- #define PHY FULLDUPLEX 10M ((uint16 t)0x0100U)
- #define PHY HALFDUPLEX 10M ((uint16 t)0x0000U)
- #define PHY AUTONEGOTIATION ((uint16 t)0x1000U)
- #define PHY\_RESTART\_AUTONEGOTIATION ((uint16\_t)0x0200U)
- #define PHY POWERDOWN ((uint16 t)0x0800U)
- #define PHY ISOLATE ((uint16 t)0x0400U)
- #define PHY AUTONEGO COMPLETE ((uint16 t)0x0020U)
- #define PHY LINKED STATUS ((uint16 t)0x0004U)
- #define PHY\_JABBER\_DETECTION ((uint16\_t)0x0002U)
- #define PHY\_SR ((uint16\_t)0x10U)
- #define PHY\_SPEED\_STATUS ((uint16\_t)0x0002U)
- #define PHY\_DUPLEX\_STATUS ((uint16\_t)0x0004U)
- #define assert\_param(expr) ((expr) ? (void)0U : assert\_failed((uint8\_t \*)\_\_FILE\_\_, \_\_LINE\_\_))

#### **Functions**

• void assert\_failed (uint8\_t \*file, uint32\_t line)

#### 13.5.1 Detailed Description

Attention

#### © COPYRIGHT(c) 2019 STMicroelectronics

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of STMicroelectronics nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

#### 13.5.2 Macro Definition Documentation

13.5.2.1 #define assert\_param(\_expr\_) ((expr) ? (void)OU : assert\_failed((uint8\_t\*)\_FILE\_\_, \_LINE\_\_))

Include module's header file.

The assert param macro is used for function's parameters check.

#### **Parameters**

expr

If expr is false, it calls assert\_failed function which reports the name of the source file and the source line number of the call that failed. If expr is true, it returns no value.

#### Return values

None

#### 13.5.2.2 #define EXTERNAL\_CLOCK\_VALUE ((uint32\_t)12288000U)

External clock source for I2S peripheral This value is used by the I2S HAL module to compute the I2S clock source frequency, this source is inserted directly through I2S CKIN pad.

Value of the Internal oscillator in Hz

13.5.2.3 #define HSE\_STARTUP\_TIMEOUT ((uint32\_t)100U)

Time out for HSE start up, in ms

13.5.2.4 #define HSE VALUE ((uint32 t)8000000U)

Adjust the value of External High Speed oscillator (HSE) used in your application. This value is used by the RCC HAL module to compute the system frequency (when HSE is used as system clock source, directly or through the PLL).

Value of the External oscillator in Hz

13.5.2.5 #define HSI\_VALUE ((uint32\_t)16000000U)

Internal High Speed oscillator (HSI) value. This value is used by the RCC HAL module to compute the system frequency (when HSI is used as system clock source, directly or through the PLL).

Value of the Internal oscillator in Hz

13.5.2.6 #define LSE\_STARTUP\_TIMEOUT ((uint32\_t)5000U)

Time out for LSE start up, in ms

13.5.2.7 #define LSE\_VALUE ((uint32\_t)32768U)

External Low Speed oscillator (LSE) value.

< Value of the Internal Low Speed oscillator in Hz The real value may vary depending on the variations in voltage and temperature. Value of the External Low Speed oscillator in Hz

13.5.2.8 #define LSI\_VALUE ((uint32\_t)32000U)

Internal Low Speed oscillator (LSI) value.

LSI Typical Value in Hz

13.5.2.9 #define PHY\_AUTONEGO\_COMPLETE ((uint16\_t)0x0020U)

Auto-Negotiation process completed

13.5.2.10 #define PHY\_AUTONEGOTIATION ((uint16\_t)0x1000U)

Enable auto-negotiation function

13.5.2.11 #define PHY\_BCR ((uint16\_t)0x0000U)

Transceiver Basic Control Register

13.5.2.12 #define PHY\_BSR ((uint16\_t)0x0001U)

Transceiver Basic Status Register

13.5.2.13 #define PHY\_DUPLEX\_STATUS ((uint16\_t)0x0004U)

PHY Duplex mask

13.5.2.14 #define PHY\_FULLDUPLEX\_100M ((uint16\_t)0x2100U) Set the full-duplex mode at 100 Mb/s 13.5.2.15 #define PHY\_FULLDUPLEX\_10M ((uint16\_t)0x0100U) Set the full-duplex mode at 10 Mb/s 13.5.2.16 #define PHY\_HALFDUPLEX\_100M ((uint16\_t)0x2000U) Set the half-duplex mode at 100 Mb/s 13.5.2.17 #define PHY\_HALFDUPLEX\_10M ((uint16\_t)0x0000U) Set the half-duplex mode at 10 Mb/s 13.5.2.18 #define PHY\_ISOLATE ((uint16\_t)0x0400U) Isolate PHY from MII 13.5.2.19 #define PHY\_JABBER\_DETECTION ((uint16\_t)0x0002U) Jabber condition detected 13.5.2.20 #define PHY\_LINKED\_STATUS ((uint16\_t)0x0004U) Valid link established 13.5.2.21 #define PHY\_LOOPBACK ((uint16\_t)0x4000U) Select loop-back mode 13.5.2.22 #define PHY\_POWERDOWN ((uint16\_t)0x0800U) Select the power down mode 13.5.2.23 #define PHY\_RESET ((uint16\_t)0x8000U)

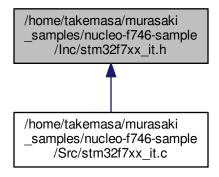
**PHY Reset** 

13.5.2.24 #define PHY\_RESTART\_AUTONEGOTIATION ((uint16\_t)0x0200U) Restart auto-negotiation function 13.5.2.25 #define PHY\_SPEED\_STATUS ((uint16\_t)0x0002U) PHY Speed mask 13.5.2.26 #define PHY\_SR ((uint16\_t)0x10U) PHY status register Offset 13.5.2.27 #define TICK\_INT\_PRIORITY ((uint32\_t)0U) tick interrupt priority 13.5.2.28 #define VDD\_VALUE ((uint32\_t)3300U) This is the HAL system configuration section. Value of VDD in mv 13.5.3 Function Documentation 13.5.3.1 void assert\_failed ( uint8\_t \* file, uint32\_t line ) Reports the name of the source file and the source line number where the assert\_param error has occurred. **Parameters** file pointer to the source file name line assert\_param error line source number

None

## 13.6 /home/takemasa/murasaki\_samples/nucleo-f746-sample/Inc/stm32f7xx\_it.h File Reference

This graph shows which files directly or indirectly include this file:



#### 13.6.1 Detailed Description

#### COPYRIGHT(c) 2019 STMicroelectronics

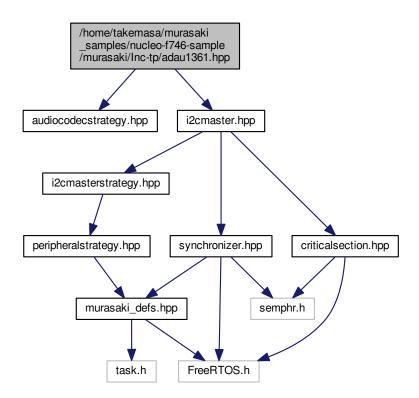
Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3. Neither the name of STMicroelectronics nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

# 13.7 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc-tp/adau1361.hpp File Reference

#include <audiocodecstrategy.hpp>
#include "i2cmaster.hpp"
Include dependency graph for adau1361.hpp:



#### Classes

· class murasaki::Adau1361

#### **Namespaces**

• murasaki

## 13.7.1 Detailed Description

Date

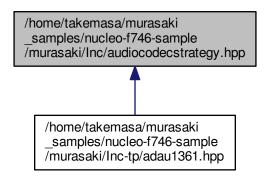
2018/05/11

Author

: takemasa

# 13.8 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/audiocodecstrategy.hpp File Reference

This graph shows which files directly or indirectly include this file:



## Classes

• class murasaki::AudioCodecStrategy

## **Namespaces**

• murasaki

## 13.8.1 Detailed Description

Date

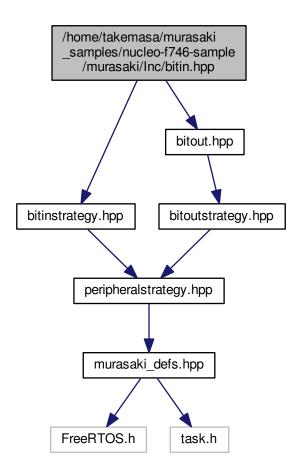
2018/05/11

Author

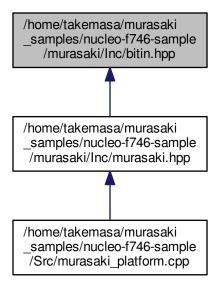
: takemasa

# 13.9 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/bitin.hpp File Reference

#include <bitinstrategy.hpp>
#include "bitout.hpp"
Include dependency graph for bitin.hpp:



This graph shows which files directly or indirectly include this file:



#### **Classes**

· class murasaki::BitIn

## **Namespaces**

· murasaki

## 13.9.1 Detailed Description

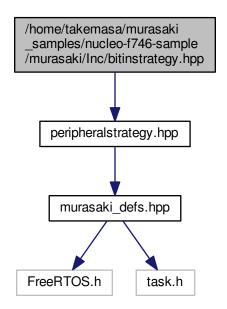
Date

2018/05/07

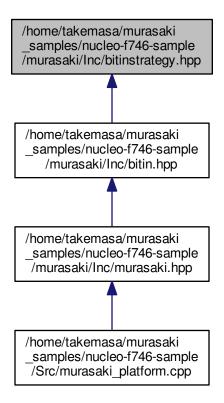
Author

13.10 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/bitinstrategy.hpp File Reference

#include <peripheralstrategy.hpp>
Include dependency graph for bitinstrategy.hpp:



This graph shows which files directly or indirectly include this file:



## **Classes**

· class murasaki::BitInStrategy

## **Namespaces**

• murasaki

## 13.10.1 Detailed Description

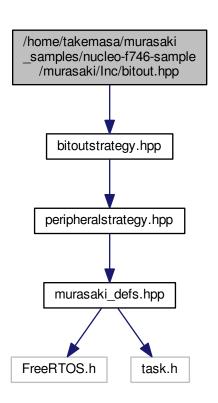
Date

2018/05/07

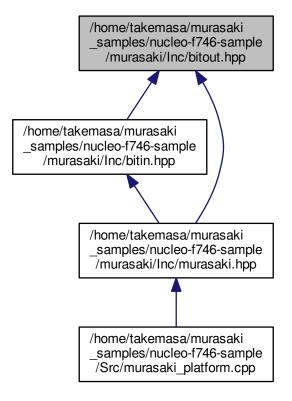
Author

#### /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/bitout.hpp 13.11 File Reference

#include <bitoutstrategy.hpp> Include dependency graph for bitout.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

- struct murasaki::GPIO\_type
- · class murasaki::BitOut

#### **Namespaces**

• murasaki

## 13.11.1 Detailed Description

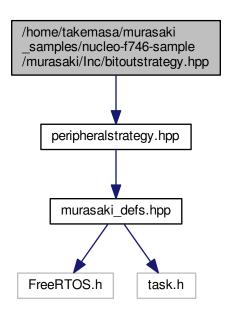
Date

2018/05/07

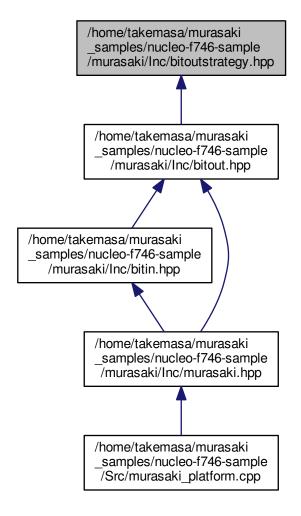
Author

13.12 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/bitoutstrategy.hpp File Reference

#include <peripheralstrategy.hpp>
Include dependency graph for bitoutstrategy.hpp:



This graph shows which files directly or indirectly include this file:



#### **Classes**

· class murasaki::BitOutStrategy

## **Namespaces**

murasaki

## 13.12.1 Detailed Description

Date

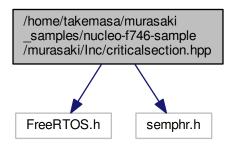
2018/05/07

Author

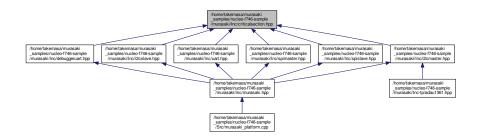
## 13.13 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/criticalsection.hpp File Reference

#include <FreeRTOS.h>
#include <semphr.h>

Include dependency graph for criticalsection.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

• class murasaki::CriticalSection

## **Namespaces**

• murasaki

## 13.13.1 Detailed Description

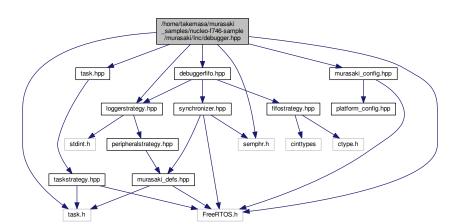
Date

2018/01/27

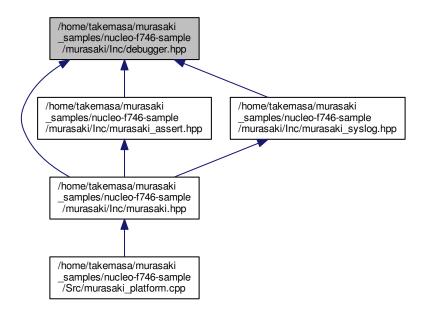
Author

## 13.14 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/debugger.hpp File Reference

```
#include <FreeRTOS.h>
#include <loggerstrategy.hpp>
#include <task.h>
#include <semphr.h>
#include "murasaki_config.hpp"
#include "debuggerfifo.hpp"
#include "task.hpp"
Include dependency graph for debugger.hpp:
```



This graph shows which files directly or indirectly include this file:



#### Classes

· class murasaki::Debugger

#### **Namespaces**

· murasaki

#### **Variables**

• Debugger \* murasaki::debugger

## 13.14.1 Detailed Description

Date

2018/01/03

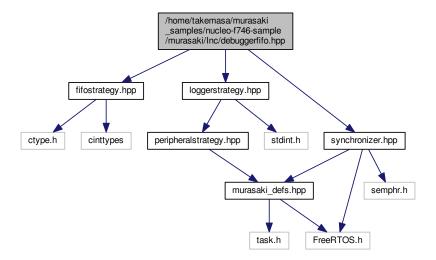
Author

takemasa

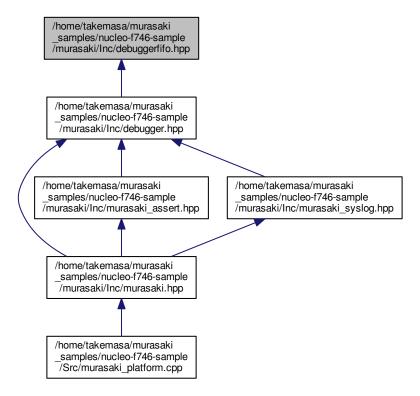
This class serves printf function for both task context and ISR context.

# 13.15 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/debuggerfifo.hpp File Reference

```
#include <fifostrategy.hpp>
#include <loggerstrategy.hpp>
#include "synchronizer.hpp"
Include dependency graph for debuggerfifo.hpp:
```



This graph shows which files directly or indirectly include this file:



#### Classes

- · class murasaki::DebuggerFifo
- struct murasaki::LoggingHelpers

## **Namespaces**

• murasaki

## 13.15.1 Detailed Description

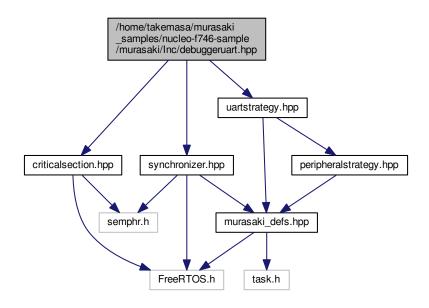
Date

2018/03/01

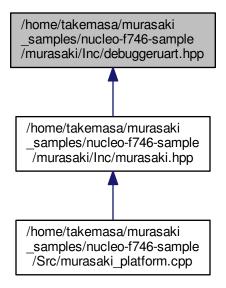
Author

13.16 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/debuggeruart.hpp File Reference

```
#include <synchronizer.hpp>
#include <uartstrategy.hpp>
#include "criticalsection.hpp"
Include dependency graph for debuggeruart.hpp:
```



This graph shows which files directly or indirectly include this file:



#### **Classes**

• class murasaki::DebuggerUart

## **Namespaces**

· murasaki

## 13.16.1 Detailed Description

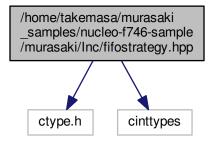
Date

2018/09/23

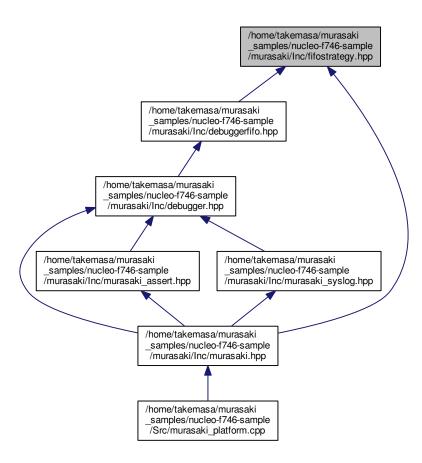
Author

## 13.17 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/fifostrategy.hpp File Reference

#include <ctype.h>
#include <cinttypes>
Include dependency graph for fifostrategy.hpp:



This graph shows which files directly or indirectly include this file:



#### **Classes**

· class murasaki::FifoStrategy

## **Namespaces**

• murasaki

## 13.17.1 Detailed Description

Date

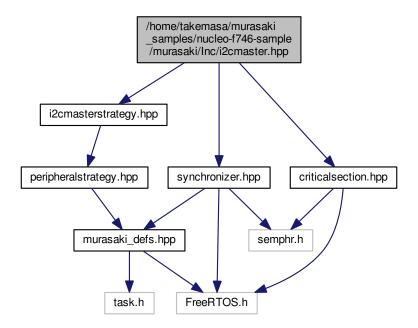
2018/02/26

**Author** 

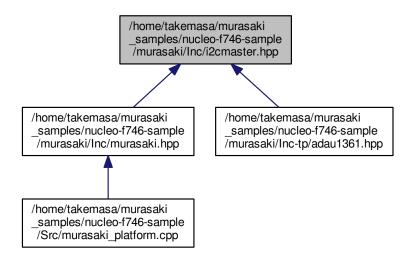
takemasa

# 13.18 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/i2cmaster.hpp File Reference

```
#include <i2cmasterstrategy.hpp>
#include <synchronizer.hpp>
#include "criticalsection.hpp"
Include dependency graph for i2cmaster.hpp:
```



This graph shows which files directly or indirectly include this file:



## **Classes**

· class murasaki::I2cMaster

## **Namespaces**

• murasaki

## 13.18.1 Detailed Description

Date

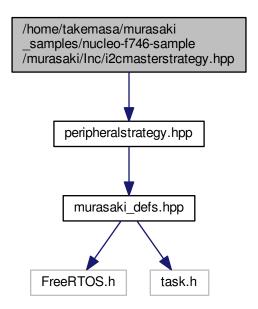
2018/02/12

Author

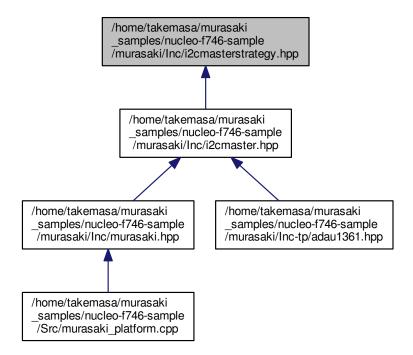
: takemasa

13.19 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/i2cmasterstrategy.hpp File Reference

#include <peripheralstrategy.hpp>
Include dependency graph for i2cmasterstrategy.hpp:



This graph shows which files directly or indirectly include this file:



## **Classes**

· class murasaki::I2CMasterStrategy

## **Namespaces**

murasaki

## 13.19.1 Detailed Description

Date

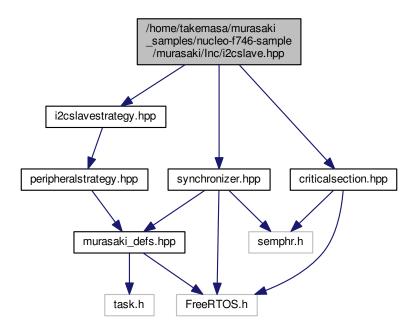
2018/02/11

#### Author

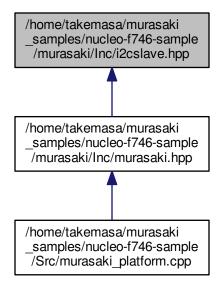
: takemasa

# 13.20 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/i2cslave.hpp File Reference

#include <i2cslavestrategy.hpp>
#include <synchronizer.hpp>
#include "criticalsection.hpp"
Include dependency graph for i2cslave.hpp:



This graph shows which files directly or indirectly include this file:



#### **Classes**

• class murasaki::I2cSlave

#### **Namespaces**

· murasaki

## 13.20.1 Detailed Description

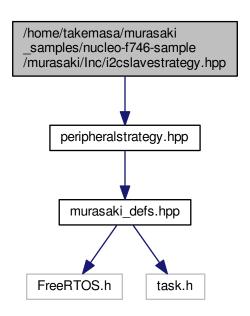
Date

2018/10/07

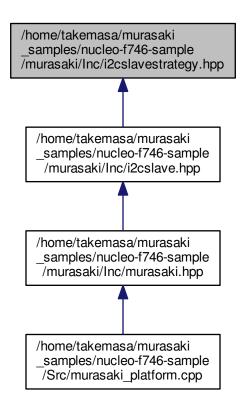
Author

13.21 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/i2cslavestrategy.hpp File Reference

#include <peripheralstrategy.hpp>
Include dependency graph for i2cslavestrategy.hpp:



This graph shows which files directly or indirectly include this file:



## **Classes**

· class murasaki::I2cSlaveStrategy

## **Namespaces**

• murasaki

## 13.21.1 Detailed Description

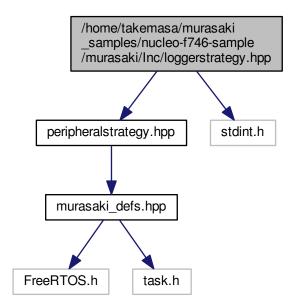
Date

2018/10/07

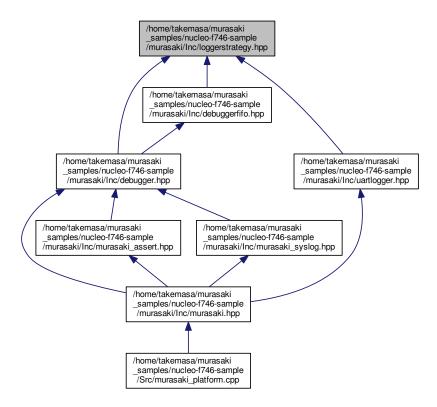
Author

13.22 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/loggerstrategy.hpp File Reference

#include <peripheralstrategy.hpp>
#include <stdint.h>
Include dependency graph for loggerstrategy.hpp:



This graph shows which files directly or indirectly include this file:



#### **Classes**

· class murasaki::LoggerStrategy

#### **Namespaces**

• murasaki

#### 13.22.1 Detailed Description

Date

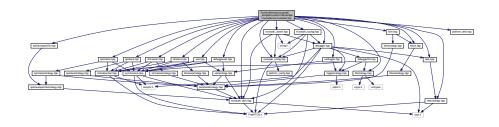
2018/01/20

Author

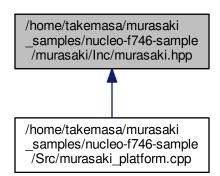
: takemasa

## 13.23 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/murasaki.hpp File Reference

```
#include <debugger.hpp>
#include <fifostrategy.hpp>
#include <taskstrategy.hpp>
#include "murasaki_config.hpp"
#include "murasaki_defs.hpp"
#include "task.hpp"
#include "uart.hpp"
#include "debuggeruart.hpp"
#include "spimaster.hpp"
#include "spislave.hpp"
#include "spislavespecifier.hpp"
#include "i2cmaster.hpp"
#include "i2cslave.hpp"
#include "bitin.hpp"
#include "bitout.hpp"
#include "uartlogger.hpp"
#include "murasaki_assert.hpp"
#include "murasaki_syslog.hpp"
#include "platform_defs.hpp"
Include dependency graph for murasaki.hpp:
```



This graph shows which files directly or indirectly include this file:



110101011	10
13.23.1	Detailed Description
Date	
20	18/01/21
Author	
	kemasa
Applicati	ion can include only this file. Other essential header files are automatically included from this file.
13.24	/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/murasaki_ 
13.24.1	Detailed Description
Date	
20	18/02/01
Author	kemasa
tar	Cinasa
13.25	/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/murasaki_ 1_env.hpp File Reference
13.25.1	Detailed Description
Date	
20	18/02/01
Author	
tak	kemasa
13.26	/home/takemasa/murasaki_samples/nucleo-f746-sample/murasaki/lnc/murasaki_ 2_ug.hpp File Reference
13.26.1	Detailed Description
Date	
20	18/02/01
Author	
tak	kemasa

# 13.27 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/murasaki\_ 3\_pg.hpp File Reference

## 13.27.1 Detailed Description

Date

May 25, 2018

**Author** 

takemasa

# 13.28 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/murasaki\_← 4\_mod.hpp File Reference

## 13.28.1 Detailed Description

Date

May 25, 2018

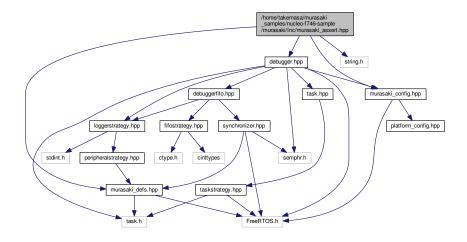
Author

takemasa

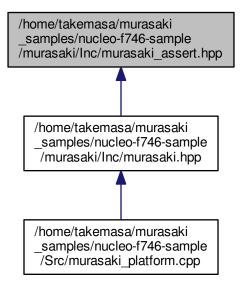
# 13.29 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/murasaki\_← assert.hpp File Reference

```
#include <debugger.hpp>
#include "murasaki_config.hpp"
#include "murasaki_defs.hpp"
#include <string.h>
```

Include dependency graph for murasaki\_assert.hpp:



This graph shows which files directly or indirectly include this file:



#### **Namespaces**

• murasaki

# Macros

- #define MURASAKI\_ASSERT(COND)
- #define MURASAKI\_PRINT\_ERROR(ERR)

## 13.29.1 Detailed Description

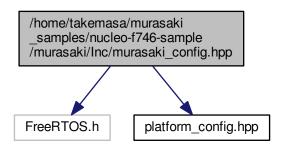
Date

2018/01/31

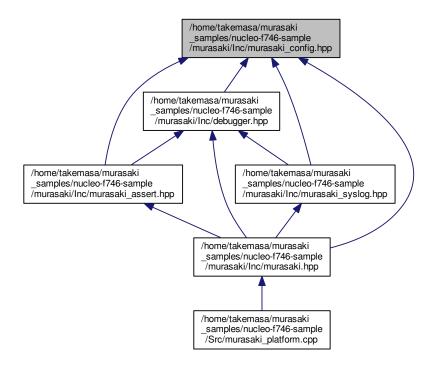
Author

# 13.30 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/murasaki\_← config.hpp File Reference

#include <FreeRTOS.h>
#include <platform\_config.hpp>
Include dependency graph for murasaki\_config.hpp:



This graph shows which files directly or indirectly include this file:



#### **Macros**

• #define PLATFORM\_CONFIG\_DEBUG\_LINE\_SIZE 256

- #define PLATFORM\_CONFIG\_DEBUG\_BUFFER\_SIZE 4096
- #define PLATFORM\_CONFIG\_DEBUG\_SERIAL\_TIMEOUT (murasaki::kwmsIndefinitely)
- #define PLATFORM\_CONFIG\_DEBUG\_TASK\_STACK\_SIZE 256
- #define PLATFORM\_CONFIG\_DEBUG\_TASK\_PRIORITY (( configMAX\_PRIORITIES-1 > 0 ) ? configM
   AX PRIORITIES-1 : 0)
- #define MURASAKI\_CONFIG\_NODEBUG false

#### 13.30.1 Detailed Description

Date

2018/01/03

**Author** 

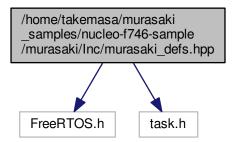
takemasa

To override the configuration, define the same name macro inside application\_config.hpp

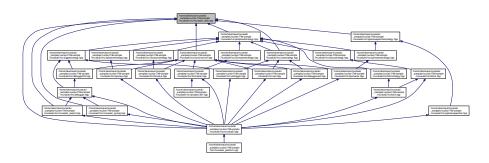
# 13.31 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/murasaki\_← defs.hpp File Reference

#include <FreeRTOS.h>
#include <task.h>

Include dependency graph for murasaki\_defs.hpp:



This graph shows which files directly or indirectly include this file:



## **Namespaces**

· murasaki

# 13.31.1 Detailed Description

Date

2017/11/05

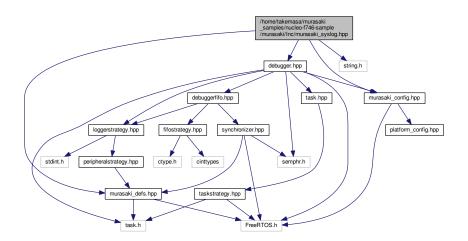
Author

takemasa

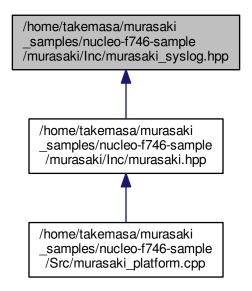
# 13.32 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/murasaki\_← syslog.hpp File Reference

```
#include <debugger.hpp>
#include "murasaki_config.hpp"
#include "murasaki_defs.hpp"
#include "string.h"
```

Include dependency graph for murasaki\_syslog.hpp:



This graph shows which files directly or indirectly include this file:



## **Namespaces**

· murasaki

#### **Macros**

• #define MURASAKI\_SYSLOG(FACILITY, SEVERITY, FORMAT, ...)

### **Functions**

- void murasaki::SetSyslogSererityThreshold (murasaki::SyslogSeverity severity)
- void murasaki::SetSyslogFacilityMask (uint32 t mask)
- void murasaki::AddSyslogFacilityToMask (murasaki::SyslogFacility facility)
- void murasaki::RemoveSyslogFacilityFromMask (murasaki::SyslogFacility facility)
- bool murasaki::AllowedSyslogOut (murasaki::SyslogFacility facility, murasaki::SyslogSeverity severity)

#### 13.32.1 Detailed Description

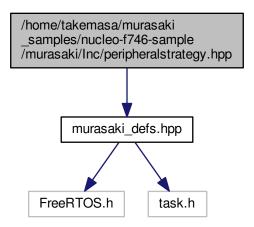
Date

2018/09/01

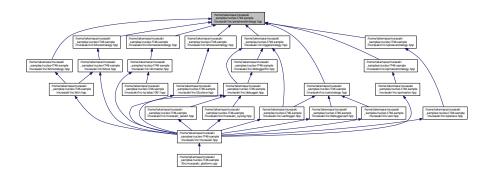
Author

# 13.33 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/peripheralstrategy.hpp File Reference

#include "murasaki\_defs.hpp"
Include dependency graph for peripheralstrategy.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

• class murasaki::PeripheralStrategy

## **Namespaces**

• murasaki

## 13.33.1 Detailed Description

Date

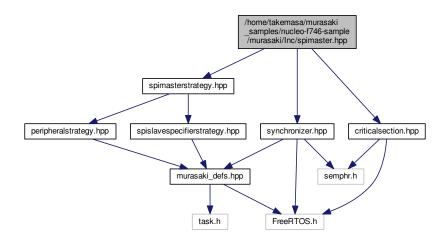
2018/04/26

#### Author

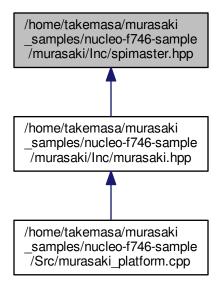
: takemasa

# 13.34 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/spimaster.hpp File Reference

```
#include <spimasterstrategy.hpp>
#include <synchronizer.hpp>
#include "criticalsection.hpp"
Include dependency graph for spimaster.hpp:
```



This graph shows which files directly or indirectly include this file:



#### **Classes**

• class murasaki::SpiMaster

## **Namespaces**

· murasaki

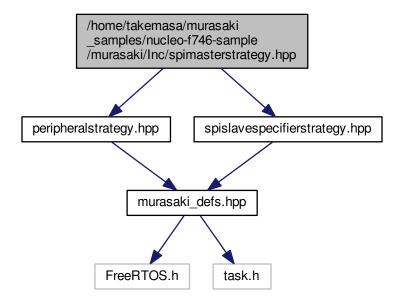
# 13.34.1 Detailed Description

Date

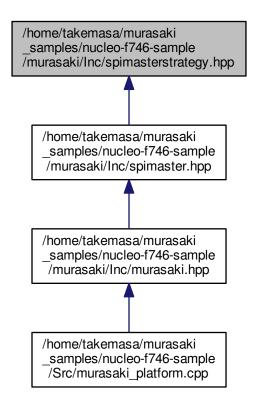
2018/02/14

Author

#include <peripheralstrategy.hpp>
#include <spislavespecifierstrategy.hpp>
Include dependency graph for spimasterstrategy.hpp:



This graph shows which files directly or indirectly include this file:



## **Classes**

· class murasaki::SpiMasterStrategy

## **Namespaces**

• murasaki

## 13.35.1 Detailed Description

Date

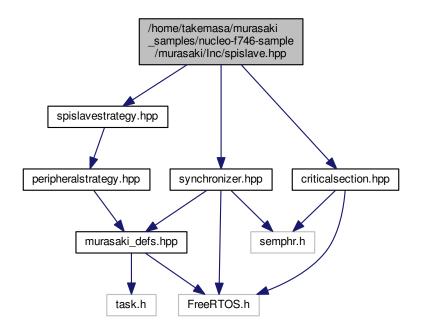
2018/02/11

Author

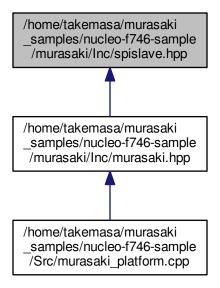
: takemasa

# 13.36 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/spislave.hpp File Reference

```
#include <spislavestrategy.hpp>
#include <synchronizer.hpp>
#include "criticalsection.hpp"
Include dependency graph for spislave.hpp:
```



This graph shows which files directly or indirectly include this file:



#### **Classes**

• class murasaki::SpiSlave

## **Namespaces**

· murasaki

# 13.36.1 Detailed Description

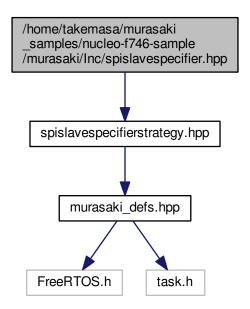
Date

2018/02/14

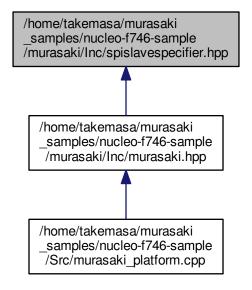
Author

13.37 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/spislavespecifier.hpp File Reference

#include <spislavespecifierstrategy.hpp>
Include dependency graph for spislavespecifier.hpp:



This graph shows which files directly or indirectly include this file:



#### **Classes**

• class murasaki::SpiSlaveSpecifier

## **Namespaces**

· murasaki

# 13.37.1 Detailed Description

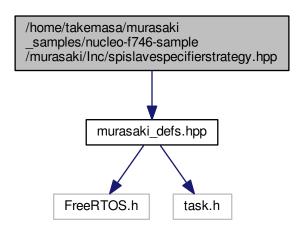
Date

2018/02/17

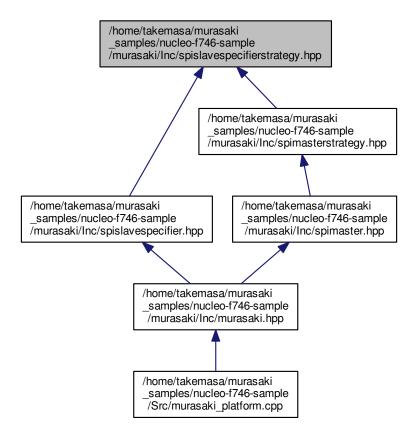
Author

13.38 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/spislavespecifierstrategy.h File Reference

#include "murasaki\_defs.hpp"
Include dependency graph for spislavespecifierstrategy.hpp:



This graph shows which files directly or indirectly include this file:



#### **Classes**

• class murasaki::SpiSlaveSpecifierStrategy

## **Namespaces**

· murasaki

## 13.38.1 Detailed Description

Date

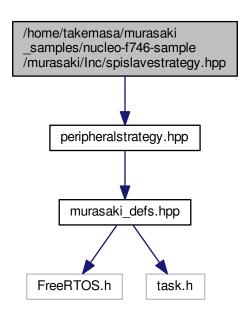
2018/02/11

Author

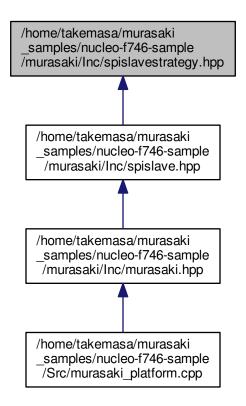
: takemasa

13.39 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/spislavestrategy.hpp File Reference

#include <peripheralstrategy.hpp>
Include dependency graph for spislavestrategy.hpp:



This graph shows which files directly or indirectly include this file:



## **Classes**

· class murasaki::SpiSlaveStrategy

## **Namespaces**

• murasaki

# 13.39.1 Detailed Description

Date

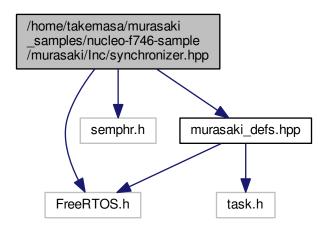
2018/02/11

Author

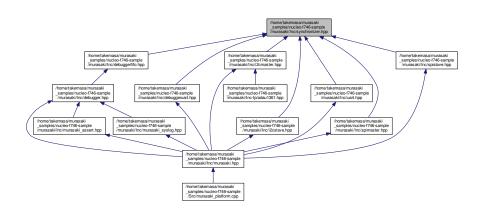
: takemasa

# 13.40 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/synchronizer.hpp File Reference

```
#include <FreeRTOS.h>
#include <semphr.h>
#include <murasaki_defs.hpp>
Include dependency graph for synchronizer.hpp:
```



This graph shows which files directly or indirectly include this file:



#### **Classes**

• class murasaki::Synchronizer

## **Namespaces**

• murasaki

# 13.40.1 Detailed Description

Date

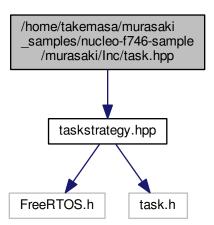
2018/01/26

**Author** 

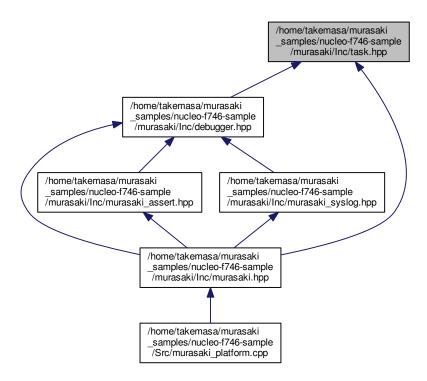
takemasa

# 13.41 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/task.hpp File Reference

#include <taskstrategy.hpp>
Include dependency graph for task.hpp:



This graph shows which files directly or indirectly include this file:



#### **Classes**

· class murasaki::Task

## **Namespaces**

· murasaki

## 13.41.1 Detailed Description

Date

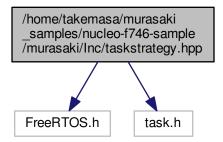
2019/02/03

Author

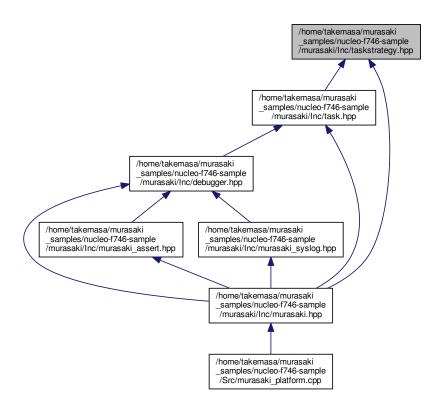
# 13.42 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/taskstrategy.hpp File Reference

#include <FreeRTOS.h>
#include <task.h>

Include dependency graph for taskstrategy.hpp:



This graph shows which files directly or indirectly include this file:



## Classes

· class murasaki::TaskStrategy

## **Namespaces**

• murasaki

## 13.42.1 Detailed Description

Date

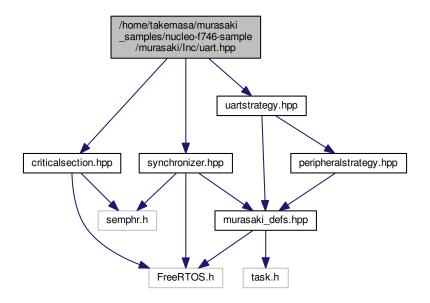
2018/02/20

#### **Author**

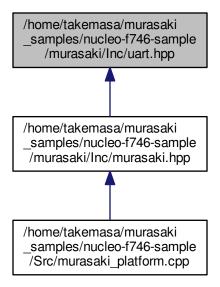
: takemasa

# 13.43 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/uart.hpp File Reference

```
#include <synchronizer.hpp>
#include <uartstrategy.hpp>
#include "criticalsection.hpp"
Include dependency graph for uart.hpp:
```



This graph shows which files directly or indirectly include this file:



#### **Classes**

· class murasaki::Uart

## **Namespaces**

· murasaki

# 13.43.1 Detailed Description

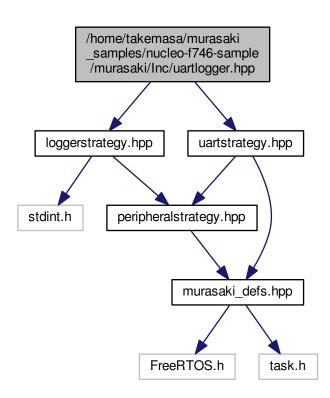
Date

2017/11/05

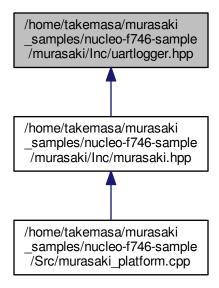
Author

# 13.44 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/uartlogger.hpp File Reference

#include <loggerstrategy.hpp>
#include <uartstrategy.hpp>
Include dependency graph for uartlogger.hpp:



This graph shows which files directly or indirectly include this file:



#### **Classes**

· class murasaki::UartLogger

## **Namespaces**

· murasaki

# 13.44.1 Detailed Description

Date

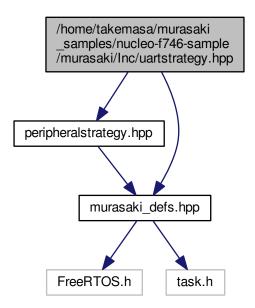
2018/01/20

Author

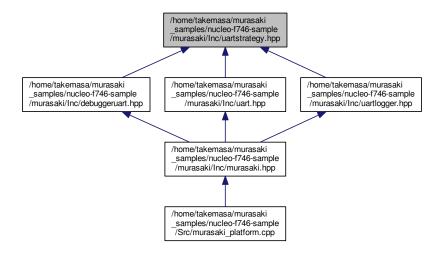
: takemasa

# 13.45 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/lnc/uartstrategy.hpp File Reference

#include <peripheralstrategy.hpp>
#include "murasaki\_defs.hpp"
Include dependency graph for uartstrategy.hpp:



This graph shows which files directly or indirectly include this file:



#### Classes

· class murasaki::UartStrategy

## **Namespaces**

· murasaki

## 13.45.1 Detailed Description

Date

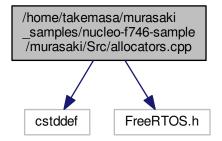
2017/11/04

#### **Author**

: Takemasa Nakamura

# 13.46 /home/takemasa/murasaki\_samples/nucleo-f746-sample/murasaki/Src/allocators.cpp File Reference

```
#include <cstddef>
#include <FreeRTOS.h>
Include dependency graph for allocators.cpp:
```



#### **Functions**

- void \* operator new (std::size\_t size)
- void \* operator new[] (std::size\_t size)
- void operator delete (void \*ptr)
- void operator delete[] (void \*ptr)

## 13.46.1 Detailed Description

Date

2018/05/02

**Author** 

takemasa

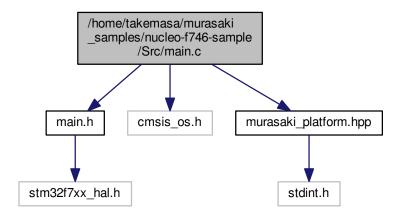
These definitions allows to used the FreeRTOS's heap instead of the system heap.

The system heap by the standard library doesn't check the limit of the heap cerefly. As a result, it is not clear how to detect the over committing memory.

FreeRTOS hepa is considered safer than system heap. Then, the new and the delete operators are overloaded to use the pvPortMalloc().

# 13.47 /home/takemasa/murasaki\_samples/nucleo-f746-sample/Src/main.c File Reference

```
#include "main.h"
#include "cmsis_os.h"
#include "murasaki_platform.hpp"
Include dependency graph for main.c:
```



#### **Functions**

- void SystemClock Config (void)
- void StartDefaultTask (void const \*argument)
- int main (void)
- void HAL\_TIM\_PeriodElapsedCallback (TIM\_HandleTypeDef \*htim)
- void Error Handler (void)
- void assert\_failed (uint8\_t \*file, uint32\_t line)

#### **Variables**

DMA\_HandleTypeDef hdma\_spi1\_tx

#### 13.47.1 Detailed Description

This notice applies to any and all portions of this file that are not between comment pairs USER CODE BEGIN and USER CODE END. Other portions of this file, whether inserted by the user or by software development tools are owned by their respective copyright owners.

Copyright (c) 2019 STMicroelectronics International N.V. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted, provided that the following conditions are met:

- Redistribution of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3. Neither the name of STMicroelectronics nor the names of other contributors to this software may be used to endorse or promote products derived from this software without specific written permission.
- 4. This software, including modifications and/or derivative works of this software, must execute solely and exclusively on microcontroller or microprocessor devices manufactured by or for STMicroelectronics.
- 5. Redistribution and use of this software other than as permitted under this license is void and will automatically terminate your rights under this license.

THIS SOFTWARE IS PROVIDED BY STMICROELECTRONICS AND CONTRIBUTORS "AS IS" AND ANY  $E \leftarrow XPRESS$ , IMPLIED OR STATUTORY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED  $W \leftarrow ARRANTIES$  OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS ARE DISCLAIMED TO THE FULLEST EXTENT PER  $\leftarrow MITTED$  BY LAW. IN NO EVENT SHALL STMICROELECTRONICS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, B  $\leftarrow W$  UT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WH  $\leftarrow W$  ETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

#### 13.47.2 Function Documentation

13.47.2.1 void assert\_failed ( uint8\_t \* file, uint32\_t line )

Reports the name of the source file and the source line number where the assert param error has occurred.

#### **Parameters**

file	pointer to the source file name
line	assert_param error line source number

Return values				
None				
13.47.2.2 void Error_Handler ( void )				
This function is executed in case of error occurrence.				
Return values				
None				
13.47.2.3 void HAL_TIM_PeriodElapsedCallback ( TIM_HandleTypeDef * htim )				
Period elapsed callback in non blocking mode.				
Toriod Stapecd Sanbask in Horr bissking mode.				
Note				
This function is called when TIM14 interrupt took place, inside HAL_TIM_IRQHandler(). It makes a direct call to HAL_IncTick() to increment a global variable "uwTick" used as application time base.				
Parameters				
htim : TIM handle				
Return values				
None				
13.47.2.4 int main ( void )				
The application entry point.				
Return values				
int				
13.47.2.5 void StartDefaultTask ( void const * argument )				
Function implementing the defaultTask thread.				
Parameters				
argument Not used				
· · · · · · · · · · · · · · · · · · ·				

		lues

None	
------	--

13.47.2.6 void SystemClock\_Config (void)

System Clock Configuration.

Return values

None

Configure LSE Drive Capability

Configure the main internal regulator output voltage

Initializes the CPU, AHB and APB busses clocks

Initializes the CPU, AHB and APB busses clocks

#### 13.47.3 Variable Documentation

13.47.3.1 DMA\_HandleTypeDef hdma\_spi1\_tx

File Name: stm32f7xx\_hal\_msp.c Description: This file provides code for the MSP Initialization and de-Initialization codes.

This notice applies to any and all portions of this file that are not between comment pairs USER CODE BEGIN and USER CODE END. Other portions of this file, whether inserted by the user or by software development tools are owned by their respective copyright owners.

Copyright (c) 2019 STMicroelectronics International N.V. All rights reserved.

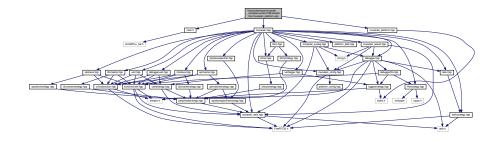
Redistribution and use in source and binary forms, with or without modification, are permitted, provided that the following conditions are met:

- Redistribution of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3. Neither the name of STMicroelectronics nor the names of other contributors to this software may be used to endorse or promote products derived from this software without specific written permission.
- 4. This software, including modifications and/or derivative works of this software, must execute solely and exclusively on microcontroller or microprocessor devices manufactured by or for STMicroelectronics.
- 5. Redistribution and use of this software other than as permitted under this license is void and will automatically terminate your rights under this license.

THIS SOFTWARE IS PROVIDED BY STMICROELECTRONICS AND CONTRIBUTORS "AS IS" AND ANY  $E \leftarrow XPRESS$ , IMPLIED OR STATUTORY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED  $W \leftarrow ARRANTIES$  OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS ARE DISCLAIMED TO THE FULLEST EXTENT PER  $\leftarrow MITTED$  BY LAW. IN NO EVENT SHALL STMICROELECTRONICS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, B  $\leftarrow W$  UT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WH  $\leftarrow W$  ETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

# 13.48 /home/takemasa/murasaki\_samples/nucleo-f746-sample/Src/murasaki\_platform.cpp

```
#include <murasaki_platform.hpp>
#include "main.h"
#include "murasaki.hpp"
Include dependency graph for murasaki platform.cpp:
```



#### **Functions**

- void InitPlatform ()
- void ExecPlatform ()
- void HAL UART TxCpltCallback (UART HandleTypeDef \*huart)
- void HAL UART RxCpltCallback (UART HandleTypeDef \*huart)
- void HAL UART ErrorCallback (UART HandleTypeDef \*huart)
- void HAL\_SPI\_TxRxCpltCallback (SPI\_HandleTypeDef \*hspi)
- void HAL SPI ErrorCallback (SPI HandleTypeDef \*hspi)
- void HAL\_I2C\_MasterTxCpltCallback (I2C\_HandleTypeDef \*hi2c)
- void HAL I2C MasterRxCpltCallback (I2C HandleTypeDef \*hi2c)
- void HAL I2C SlaveTxCpltCallback (I2C HandleTypeDef \*hi2c)
- void HAL\_I2C\_SlaveRxCpltCallback (I2C\_HandleTypeDef \*hi2c)
- void HAL\_I2C\_ErrorCallback (I2C\_HandleTypeDef \*hi2c)
- void HAL\_GPIO\_EXTI\_Callback (uint16\_t GPIO\_P)
- void CustomAssertFailed (uint8 t \*file, uint32 t line)
- void CustomDefaultHandler ()

## 13.48.1 Detailed Description

Date

2018/05/20

**Author** 

takemasa

#### 13.48.2 Function Documentation

13.48.2.1 void HAL\_I2C\_MasterRxCpltCallback ( I2C\_HandleTypeDef \* hi2c )

Essential to sync up with I2C.

**Parameters** 

hi2c

This is called from inside of HAL when an I2C receive done interrupt is accepted.

STM32Cube HAL has same name function internally. That function is invoked whenever an relevant interrupt happens. In the other hand, that function is declared as weak bound. As a result, this function overrides the default RX interrupt call back.

In this call back, the uart device handle have to be passed to the murasaki::Uart::ReceiveCompleteCallback() function.

13.48.2.2 void HAL\_I2C\_SlaveRxCpltCallback ( I2C\_HandleTypeDef \* hi2c )

Essential to sync up with I2C.

**Parameters** 

hi2c

This is called from inside of HAL when an I2C receive done interrupt is accepted.

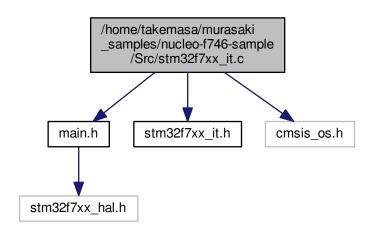
STM32Cube HAL has same name function internally. That function is invoked whenever an relevant interrupt happens. In the other hand, that function is declared as weak bound. As a result, this function overrides the default RX interrupt call back.

In this call back, the I2C slave device handle have to be passed to the murasaki::I2cSlave::ReceiveComplete Callback() function.

# 13.49 /home/takemasa/murasaki\_samples/nucleo-f746-sample/Src/stm32f7xx\_it.c File Reference

```
#include "main.h"
#include "stm32f7xx_it.h"
#include "cmsis_os.h"
```

Include dependency graph for stm32f7xx it.c:



#### **Variables**

DMA HandleTypeDef hdma spi1 tx

#### 13.49.1 Detailed Description

#### COPYRIGHT(c) 2019 STMicroelectronics

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3. Neither the name of STMicroelectronics nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

232 File Documentation

#### 13.49.2 Variable Documentation

13.49.2.1 DMA\_HandleTypeDef hdma\_spi1\_tx

File Name: stm32f7xx\_hal\_msp.c Description: This file provides code for the MSP Initialization and de-Initialization codes.

This notice applies to any and all portions of this file that are not between comment pairs USER CODE BEGIN and USER CODE END. Other portions of this file, whether inserted by the user or by software development tools are owned by their respective copyright owners.

Copyright (c) 2019 STMicroelectronics International N.V. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted, provided that the following conditions are met:

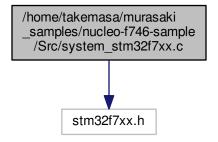
- Redistribution of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3. Neither the name of STMicroelectronics nor the names of other contributors to this software may be used to endorse or promote products derived from this software without specific written permission.
- 4. This software, including modifications and/or derivative works of this software, must execute solely and exclusively on microcontroller or microprocessor devices manufactured by or for STMicroelectronics.
- 5. Redistribution and use of this software other than as permitted under this license is void and will automatically terminate your rights under this license.

THIS SOFTWARE IS PROVIDED BY STMICROELECTRONICS AND CONTRIBUTORS "AS IS" AND ANY E  $\leftarrow$  XPRESS, IMPLIED OR STATUTORY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED W  $\leftarrow$  ARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS ARE DISCLAIMED TO THE FULLEST EXTENT PER  $\leftarrow$  MITTED BY LAW. IN NO EVENT SHALL STMICROELECTRONICS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, B  $\leftarrow$  UT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WH  $\leftarrow$  ETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

# 13.50 /home/takemasa/murasaki\_samples/nucleo-f746-sample/Src/system\_stm32f7xx.c File Reference

#include "stm32f7xx.h"

Include dependency graph for system\_stm32f7xx.c:



#### **Macros**

- #define HSE\_VALUE ((uint32\_t)25000000)
- #define HSI VALUE ((uint32 t)16000000)
- #define VECT\_TAB\_OFFSET 0x00

#### **Functions**

- void SystemInit (void)
- void SystemCoreClockUpdate (void)

### 13.50.1 Detailed Description

#### Author

MCD Application Team This file provides two functions and one global variable to be called from user application:

- SystemInit(): This function is called at startup just after reset and before branch to main program. This call is made inside the "startup\_stm32f7xx.s" file.
- SystemCoreClock variable: Contains the core clock (HCLK), it can be used by the user application to setup the SysTick timer or configure other parameters.
- SystemCoreClockUpdate(): Updates the variable SystemCoreClock and must be called whenever the core clock is changed during program execution.

#### Attention

234 File Documentation

#### © COPYRIGHT 2016 STMicroelectronics

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- 3. Neither the name of STMicroelectronics nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

### Index

/home/takemasa/murasaki_samples/nucleo-f746-	sample/murasaki/Inc/i2cslave.hpp, 188
sample/Inc/main.h, 155	/home/takemasa/murasaki_samples/nucleo-f746-
/home/takemasa/murasaki_samples/nucleo-f746-	sample/murasaki/Inc/i2cslavestrategy.hpp,
sample/Inc/murasaki_platform.hpp, 157	190
/home/takemasa/murasaki_samples/nucleo-f746-	/home/takemasa/murasaki_samples/nucleo-f746-
sample/Inc/platform_config.hpp, 158	sample/murasaki/Inc/loggerstrategy.hpp, 192
/home/takemasa/murasaki_samples/nucleo-f746-	/home/takemasa/murasaki_samples/nucleo-f746-
sample/Inc/platform_defs.hpp, 159	sample/murasaki/Inc/murasaki.hpp, 194
/home/takemasa/murasaki_samples/nucleo-f746-	/home/takemasa/murasaki_samples/nucleo-f746-
sample/Inc/stm32f7xx_hal_conf.h, 160	sample/murasaki/lnc/murasaki_0_intro.hpp,
/home/takemasa/murasaki_samples/nucleo-f746-	195
sample/Inc/stm32f7xx_it.h, 166	/home/takemasa/murasaki_samples/nucleo-f746-
/home/takemasa/murasaki_samples/nucleo-f746-	sample/murasaki/Inc/murasaki_1_env.hpp,
sample/Src/main.c, 225	195
/home/takemasa/murasaki_samples/nucleo-f746-	/home/takemasa/murasaki_samples/nucleo-f746-
sample/Src/murasaki_platform.cpp, 229	sample/murasaki/Inc/murasaki_2_ug.hpp,
/home/takemasa/murasaki_samples/nucleo-f746-	195
sample/Src/stm32f7xx_it.c, 231	/home/takemasa/murasaki_samples/nucleo-f746-
/home/takemasa/murasaki_samples/nucleo-f746-	sample/murasaki/Inc/murasaki_3_pg.hpp,
sample/Src/system_stm32f7xx.c, 232	196
/home/takemasa/murasaki_samples/nucleo-f746-	/home/takemasa/murasaki_samples/nucleo-f746-
sample/murasaki/Inc-tp/adau1361.hpp, 167	sample/murasaki/Inc/murasaki_4_mod.hpp,
/home/takemasa/murasaki_samples/nucleo-f746-	196
sample/murasaki/Inc/audiocodecstrategy.hpp,	/home/takemasa/murasaki_samples/nucleo-f746-
168	sample/murasaki/lnc/murasaki_assert.hpp,
/home/takemasa/murasaki_samples/nucleo-f746-	196
sample/murasaki/Inc/bitin.hpp, 169	/home/takemasa/murasaki_samples/nucleo-f746-
/home/takemasa/murasaki_samples/nucleo-f746-	sample/murasaki/lnc/murasaki_config.hpp,
sample/murasaki/Inc/bitinstrategy.hpp, 171	198
/home/takemasa/murasaki_samples/nucleo-f746-	/home/takemasa/murasaki_samples/nucleo-f746-
sample/murasaki/Inc/bitout.hpp, 173	sample/murasaki/Inc/murasaki_defs.hpp, 199
/home/takemasa/murasaki_samples/nucleo-f746-	/home/takemasa/murasaki_samples/nucleo-f746-
sample/murasaki/Inc/bitoutstrategy.hpp, 175 /home/takemasa/murasaki_samples/nucleo-f746-	sample/murasaki/lnc/murasaki_syslog.hpp, 200
sample/murasaki/Inc/criticalsection.hpp, 177	/home/takemasa/murasaki_samples/nucleo-f746-
/home/takemasa/murasaki_samples/nucleo-f746-	sample/murasaki/lnc/peripheralstrategy.hpp,
sample/murasaki/Inc/debugger.hpp, 178	202
/home/takemasa/murasaki_samples/nucleo-f746-	/home/takemasa/murasaki_samples/nucleo-f746-
sample/murasaki/Inc/debuggerfifo.hpp, 179	sample/murasaki/lnc/spimaster.hpp, 203
/home/takemasa/murasaki_samples/nucleo-f746-	/home/takemasa/murasaki_samples/nucleo-f746-
sample/murasaki/Inc/debuggeruart.hpp, 181	sample/murasaki/lnc/spimasterstrategy.hpp,
/home/takemasa/murasaki_samples/nucleo-f746-	205
sample/murasaki/Inc/fifostrategy.hpp, 183	/home/takemasa/murasaki_samples/nucleo-f746-
/home/takemasa/murasaki_samples/nucleo-f746-	sample/murasaki/lnc/spislave.hpp, 207
sample/murasaki/Inc/i2cmaster.hpp, 184	/home/takemasa/murasaki_samples/nucleo-f746-
/home/takemasa/murasaki_samples/nucleo-f746-	sample/murasaki/lnc/spislavespecifier.hpp,
sample/murasaki/Inc/i2cmasterstrategy.hpp,	209
186	/home/takemasa/murasaki_samples/nucleo-f746-
/home/takemasa/murasaki_samples/nucleo-f746-	sample/murasaki/lnc/spislavespecifierstrategy. ←

hpp, 211	murasaki::BitOut, 82
/home/takemasa/murasaki_samples/nucleo-f746-	,
sample/murasaki/Inc/spislavestrategy.hpp,	CMSIS, 57
213	configure_board
/home/takemasa/murasaki_samples/nucleo-f746-	murasaki::Adau1361, 72
sample/murasaki/Inc/synchronizer.hpp, 215	configure_pll
/home/takemasa/murasaki_samples/nucleo-f746-	murasaki::Adau1361, 72
sample/murasaki/Inc/task.hpp, 216	CustomAssertFailed
/home/takemasa/murasaki samples/nucleo-f746-	Application Specific Platform, 48
sample/murasaki/Inc/taskstrategy.hpp, 218	CustomDefaultHandler
/home/takemasa/murasaki samples/nucleo-f746-	Application Specific Platform, 48
sample/murasaki/Inc/uart.hpp, 219	,,
/home/takemasa/murasaki_samples/nucleo-f746-	DeassertCs
sample/murasaki/Inc/uartlogger.hpp, 221	murasaki::SpiSlaveSpecifier, 131
/home/takemasa/murasaki_samples/nucleo-f746-	murasaki::SpiSlaveSpecifierStrategy, 133
sample/murasaki/Inc/uartstrategy.hpp, 223	Debugger
/home/takemasa/murasaki_samples/nucleo-f746-	murasaki::Debugger, 87
sample/murasaki/Src/allocators.cpp, 224	debugger
~LoggerStrategy	Application Specific Platform, 53
murasaki::LoggerStrategy, 117	DebuggerFifo
murasakiLoggerotrategy, 117	murasaki::DebuggerFifo, 90
Abstract Classes, 54	DebuggerUart
Adau1361	murasaki::DebuggerUart, 92
murasaki::Adau1361, 72	Definitions and Configuration, 41
AddSyslogFacilityToMask	I2cStatus, 42
murasaki, 68	kfaAll, 44
AllowedSyslogOut	kfal2cMaster, 44
murasaki, 68	kfal2cSlave, 44
	kfal2s, 44
Application Specific Platform, 47	kfaKernel, 44
CustomAssertFailed, 48	kfaLog, 44
CustomDefaultHandler, 48	<del>-</del>
debugger, 53	kfaNone, 44
ExecPlatform, 49	kfaSai, 44
HAL_GPIO_EXTI_Callback, 49	kfaSerial, 44
HAL_I2C_ErrorCallback, 49	kfaSpiMaster, 44
HAL_I2C_MasterTxCpltCallback, 50	kfaSpiSlave, 44
HAL_I2C_SlaveTxCpltCallback, 50	kfaUser0, 44
HAL_SPI_ErrorCallback, 50	kfaUser1, 44
HAL_SPI_TxRxCpltCallback, 51	kfaUser2, 44
HAL_UART_ErrorCallback, 51	kfaUser3, 44
HAL_UART_RxCpltCallback, 51	kfaUser4, 44
HAL_UART_TxCpltCallback, 52	kfaUser5, 44
InitPlatform, 52	kfaUser6, 44
assert_failed	kfaUser7, 44
main.c, 226	ki2csArbitrationLost, 42
stm32f7xx_hal_conf.h, 165	ki2csBussError, 42
assert param	ki2csDMA, 42
stm32f7xx_hal_conf.h, 162	ki2csNak, 42
AssertCs	ki2csOK, 42
murasaki::SpiSlaveSpecifier, 131	ki2csOverrun, 42
murasaki::SpiSlaveSpecifierStrategy, 133	ki2csTimeOut, 42
AudioCodecStrategy	ki2csUnknown, 42
murasaki::AudioCodecStrategy, 76	kseAlert, 44
AutoRePrint	kseCritical, 44
	kseDebug, 44
murasaki::Debugger, 87	kseEmergency, 44
BitIn	kseError, 44
murasaki::BitIn, 79	kseInfomational, 44
BitOut	kseNotice, 44

kseWarning, 44	main.h, 156
ksphLatchThenShift, 43	ExecPlatform
ksphShiftThenLatch, 43	Application Specific Platform, 49
kspisAbort, 43	Application opeomer lations, 43
kspisDMA, 43	facility mask
kspisErrorFlag, 43	murasaki::Debugger, 88
•	FifoStrategy
kspisFrameError, 43	murasaki::FifoStrategy, 96
kspisModeCRC, 43	murasaki ilostrategy, 90
kspisModeFault, 43	Get
kspisOK, 43	murasaki::BitIn, 79
kspisOverflow, 43	
kspisTimeOut, 43	murasaki::BitInStrategy, 81
kspisUnknown, 43	murasaki::BitOut, 83
kspoFallThenRise, 43	murasaki::BitOutStrategy, 84
kspoRiseThenFall, 43	murasaki::DebuggerFifo, 90
kuhfcCts, 45	murasaki::FifoStrategy, 97
kuhfcCtsRts, 45	getCharacter
kuhfcNone, 45	murasaki::LoggerStrategy, 117
kuhfcRts, 45	murasaki::UartLogger, 149
kursDMA, 45	GetCpha
kursFrame, 45	murasaki::SpiSlaveSpecifierStrategy, 133
kursNoise, 45	GetCpol
kursOK, 45	murasaki::SpiSlaveSpecifierStrategy, 133
kursOverrun, 45	GetName
kursParity, 45	murasaki::TaskStrategy, 140
kursTimeOut, 45	GetPeripheralHandle
kutldleTimeout, 45	murasaki::BitIn, 79
kutNoldleTimeout, 45	murasaki::BitOut, 83
kwmsIndefinitely, 46	GetchFromTask
•	murasaki::Debugger, 87
kwmsPolling, 46	maradam 2 daggar, dr
MURASAKI_CONFIG_NODEBUG, 41	HAL_GPIO_EXTI_Callback
PLATFORM_CONFIG_DEBUG_BUFFER_SIZE,	Application Specific Platform, 49
41	HAL I2C ErrorCallback
PLATFORM_CONFIG_DEBUG_LINE_SIZE, 41	Application Specific Platform, 49
PLATFORM_CONFIG_DEBUG_SERIAL_TIME ↔	HAL_I2C_MasterRxCpltCallback
OUT, 41	murasaki_platform.cpp, 230
PLATFORM_CONFIG_DEBUG_TASK_PRIORI↔	HAL_I2C_MasterTxCpltCallback
TY, 42	
PLATFORM_CONFIG_DEBUG_TASK_STACK↔	Application Specific Platform, 50
_SIZE, 42	HAL_I2C_SlaveRxCpltCallback
SpiClockPhase, 42	murasaki_platform.cpp, 230
SpiClockPolarity, 43	HAL_I2C_SlaveTxCpltCallback
SpiStatus, 43	Application Specific Platform, 50
SyslogFacility, 43	HAL_SPI_ErrorCallback
SyslogSeverity, 44	Application Specific Platform, 50
UartHardwareFlowControl, 44	HAL_SPI_TxRxCpltCallback
UartStatus, 45	Application Specific Platform, 51
UartTimeout, 45	HAL_TIM_PeriodElapsedCallback
WaitMilliSeconds, 45	main.c, 227
DoPostMortem	HAL_UART_ErrorCallback
murasaki::LoggerStrategy, 117	Application Specific Platform, 51
murasaki::UartLogger, 148	HAL_UART_RxCpltCallback
	Application Specific Platform, 51
EXTERNAL_CLOCK_VALUE	HAL_UART_TxCpltCallback
stm32f7xx_hal_conf.h, 162	Application Specific Platform, 52
Enter	HSE_STARTUP_TIMEOUT
murasaki::CriticalSection, 85	stm32f7xx_hal_conf.h, 162
Error_Handler	HSE_VALUE
main.c, 227	STM32F7xx_System_Private_Includes, 59
•	_ ,

stm32f7xx_hal_conf.h, 162	Definitions and Configuration, 44
HSI_VALUE	kfaUser3
STM32F7xx_System_Private_Includes, 59	Definitions and Configuration, 44
stm32f7xx_hal_conf.h, 162	kfaUser4
HandleError	Definitions and Configuration, 44
murasaki::DebuggerUart, 93	kfaUser5
murasaki::I2CMasterStrategy, 105	Definitions and Configuration, 44
murasaki::I2cMaster, 101	kfaUser6
murasaki::l2cSlave, 110	Definitions and Configuration, 44
murasaki::I2cSlaveStrategy, 114	kfaUser7
murasaki::SpiMaster, 122	Definitions and Configuration, 44
murasaki::SpiMasterStrategy, 124	ki2csArbitrationLost
murasaki::SpiSlave, 127	Definitions and Configuration, 42
murasaki::SpiSlaveStrategy, 135	ki2csBussError
murasaki::Uart, 144	Definitions and Configuration, 42
murasaki::UartStrategy, 151	ki2csDMA
hdma_spi1_tx	Definitions and Configuration, 42
main.c, 228	ki2csNak
stm32f7xx_it.c, 232	Definitions and Configuration, 42
Helper classes, 55	ki2csOK
operator delete, 55	Definitions and Configuration, 42
operator delete[], 55	ki2csOverrun
operator new, 56	Definitions and Configuration, 42
operator new[], 56	ki2csTimeOut
I2cMaster	Definitions and Configuration, 42
murasaki::I2cMaster, 100	ki2csUnknown
I2cStatus	Definitions and Configuration, 42
Definitions and Configuration, 42	kseAlert
InitPlatform	Definitions and Configuration, 44
Application Specific Platform, 52	kseCritical
Application opecine Flationn, 32	Definitions and Configuration, 44
kfaAll	kseDebug
Definitions and Configuration, 44	Definitions and Configuration, 44
kfal2cMaster	kseEmergency
Definitions and Configuration, 44	Definitions and Configuration, 44
kfal2cSlave	kseError
Definitions and Configuration, 44	Definitions and Configuration, 44
kfal2s	kseInfomational
Definitions and Configuration, 44	Definitions and Configuration, 44
kfaKernel	kseNotice
Definitions and Configuration, 44	Definitions and Configuration, 44
kfaLog	kseWarning
Definitions and Configuration, 44	Definitions and Configuration, 44
kfaNone	ksphLatchThenShift
Definitions and Configuration, 44	Definitions and Configuration, 43
kfaSai	ksphShiftThenLatch
Definitions and Configuration, 44	Definitions and Configuration, 43
kfaSerial	kspisAbort
Definitions and Configuration, 44	Definitions and Configuration, 43
kfaSpiMaster	kspisDMA
Definitions and Configuration, 44	Definitions and Configuration, 43
kfaSpiSlave	kspisErrorFlag
Definitions and Configuration, 44	Definitions and Configuration, 43
kfaUser0	kspisFrameError
Definitions and Configuration, 44	Definitions and Configuration, 43
kfaUser1	kspisModeCRC
Definitions and Configuration, 44	Definitions and Configuration, 43
kfaUser2	kspisModeFault

Definitions and Configuration, 43 kspisOK	MURASAKI_CONFIG_NODEBUG Definitions and Configuration, 41
Definitions and Configuration, 43	MURASAKI_CONFIG_NOSYSLOG
kspisOverflow	platform_config.hpp, 159
Definitions and Configuration, 43	MURASAKI_PRINT_ERROR
kspisTimeOut	Murasaki Class Collection, 37
Definitions and Configuration, 43	MURASAKI_SYSLOG
kspisUnknown Definitions and Configuration, 43	Murasaki Class Collection, 37
kspoFallThenRise	main main.c, 227
Definitions and Configuration, 43	main.c
kspoRiseThenFall	assert_failed, 226
Definitions and Configuration, 43	Error_Handler, 227
kuhfcCts	HAL_TIM_PeriodElapsedCallback, 227
Definitions and Configuration, 45	hdma_spi1_tx, 228
kuhfcCtsRts	main, 227
Definitions and Configuration, 45 kuhfcNone	StartDefaultTask, 227
Definitions and Configuration, 45	SystemClock_Config, 228
kuhfcRts	main.h Error_Handler, 156
Definitions and Configuration, 45	murasaki, 67
kursDMA	AddSyslogFacilityToMask, 68
Definitions and Configuration, 45	AllowedSyslogOut, 68
kursFrame	platform, 69
Definitions and Configuration, 45	RemoveSyslogFacilityFromMask, 69
kursNoise	SetSyslogFacilityMask, 69
Definitions and Configuration, 45 kursOK	SetSyslogSererityThreshold, 69
Definitions and Configuration, 45	Murasaki Class Collection, 35
kursOverrun	MURASAKI_ASSERT, 36
Definitions and Configuration, 45	MURASAKI_PRINT_ERROR, 37
kursParity	MURASAKI_SYSLOG, 37 murasaki::Adau1361, 71
Definitions and Configuration, 45	Adau1361, 72
kursTimeOut	configure_board, 72
Definitions and Configuration, 45	configure_pll, 72
kutldleTimeout	send_command, 73
Definitions and Configuration, 45	send_command_table, 73
kutNoldleTimeout  Definitions and Configuration, 45	set_aux_input_gain, 73
kwmsIndefinitely	set_hp_output_gain, 74
Definitions and Configuration, 46	set_line_input_gain, 74
kwmsPolling	set_line_output_gain, 74
Definitions and Configuration, 46	start, 75 wait_pll_lock, 75
LOE OTABILID TIMEOUT	murasaki::AudioCodecStrategy, 75
LSE_STARTUP_TIMEOUT	AudioCodecStrategy, 76
stm32f7xx_hal_conf.h, 163 LSE VALUE	set_aux_input_gain, 76
stm32f7xx hal conf.h, 163	set_hp_output_gain, 76
LSI_VALUE	set_line_input_gain, 77
stm32f7xx_hal_conf.h, 163	set_line_output_gain, 77
Launch	set_mic_input_gain, 77
murasaki::TaskStrategy, 140	start, 77
Leave	murasaki::Bitln, 78
murasaki::CriticalSection, 85	Bitln, 79
line_	Get, 79 GetPeripheralHandle, 79
murasaki::Debugger, 88	murasaki::BitInStrategy, 80
MURASAKI ASSERT	Get, 81
Murasaki Class Collection, 36	murasaki::BitOut, 81
,	-) -

BitOut, 82	murasaki::I2cSlaveStrategy, 113
Get, 83	HandleError, 114
GetPeripheralHandle, 83	Receive, 114
Set, 83	ReceiveCompleteCallback, 114
murasaki::BitOutStrategy, 83	Transmit, 115
Get, 84	TransmitCompleteCallback, 115
Set, 85	murasaki::LoggerStrategy, 116
murasaki::CriticalSection, 85	∼LoggerStrategy, 117
Enter, 85	DoPostMortem, 117
Leave, 85	getCharacter, 117
murasaki::Debugger, 86	putMessage, 117
AutoRePrint, 87	murasaki::LoggingHelpers, 118
Debugger, 87	murasaki::PeripheralStrategy, 118
facility_mask_, 88	murasaki::Platform, 119
GetchFromTask, 87	murasaki::SpiMaster, 120
line_, 88	HandleError, 122
Printf, 87	SpiMaster, 121
RePrint, 88	TransmitAndReceive, 122
severity_, 88	TransmitAndReceiveCompleteCallback, 123
murasaki::DebuggerFifo, 89	murasaki::SpiMasterStrategy, 123
DebuggerFifo, 90	HandleError, 124
Get, 90	TransmitAndReceive, 125
SetPostMortem, 90	TransmitAndReceiveCompleteCallback, 125
murasaki::DebuggerUart, 91	murasaki::SpiSlave, 126
DebuggerUart, 92	HandleError, 127
HandleError, 93	SpiSlave, 127
	TransmitAndReceive, 128
Receive, 93	
ReceiveCompleteCallback, 94	TransmitAndReceiveCompleteCallback, 129
SetHardwareFlowControl, 94	murasaki::SpiSlaveSpecifier, 129
SetSpeed, 94	AssertCs, 131
Transmit, 95	DeassertCs, 131
TransmitCompleteCallback, 95	SpiSlaveSpecifier, 130
murasaki::FifoStrategy, 96	murasaki::SpiSlaveSpecifierStrategy, 132
FifoStrategy, 96	AssertCs, 133
Get, 97	DeassertCs, 133
Put, 97	GetCpha, 133
murasaki::GPIO_type, 97	GetCpol, 133
murasaki::I2CMasterStrategy, 104	SpiSlaveSpecifierStrategy, 132, 133
HandleError, 105	murasaki::SpiSlaveStrategy, 134
Receive, 106	HandleError, 135
ReceiveCompleteCallback, 106	TransmitAndReceive, 135
Transmit, 106	TransmitAndReceiveCompleteCallback, 136
TransmitCompleteCallback, 107	murasaki::Synchronizer, 136
TransmitThenReceive, 107	Release, 136
murasaki::I2cMaster, 98	Wait, 136
HandleError, 101	murasaki::Task, 137
I2cMaster, 100	Task, 138
Receive, 101	TaskBody, 139
ReceiveCompleteCallback, 102	murasaki::TaskStrategy, 139
Transmit, 102	GetName, 140
TransmitCompleteCallback, 103	Launch, 140
TransmitThenReceive, 103	Start, 141
murasaki::l2cSlave, 108	TaskBody, 141
HandleError, 110	TaskStrategy, 140
Receive, 110	murasaki::Uart, 141
ReceiveCompleteCallback, 111	HandleError, 144
Transmit, 111	Receive, 144
TransmitCompleteCallback, 112	ReceiveCompleteCallback, 145

SetHardwareFlowControl, 145	PHY_RESET
SetSpeed, 145	stm32f7xx_hal_conf.h, 164
Transmit, 146	PHY_RESTART_AUTONEGOTIATION
TransmitCompleteCallback, 146	stm32f7xx_hal_conf.h, 164
Uart, 143	PHY_SPEED_STATUS
murasaki::UartLogger, 147	stm32f7xx_hal_conf.h, 165
DoPostMortem, 148	PHY_SR
getCharacter, 149	stm32f7xx_hal_conf.h, 165
putMessage, 149	PLATFORM_CONFIG_DEBUG_BUFFER_SIZE
UartLogger, 148	Definitions and Configuration, 41
murasaki::UartStrategy, 149	PLATFORM_CONFIG_DEBUG_LINE_SIZE
HandleError, 151	Definitions and Configuration, 41
Receive, 151	PLATFORM_CONFIG_DEBUG_SERIAL_TIMEOUT
ReceiveCompleteCallback, 151	Definitions and Configuration, 41
SetHardwareFlowControl, 152	PLATFORM_CONFIG_DEBUG_TASK_PRIORITY
SetSpeed, 152	Definitions and Configuration, 42
Transmit, 152	PLATFORM CONFIG DEBUG TASK STACK SIZE
TransmitCompleteCallback, 153	Definitions and Configuration, 42
murasaki_platform.cpp	platform
HAL_I2C_MasterRxCpltCallback, 230	murasaki, 69
HAL_I2C_SlaveRxCpltCallback, 230	
HAL_IZO_SIAVENXOPILOAIIDACK, 230	platform_config.hpp
operator delete	MURASAKI_CONFIG_NOSYSLOG, 159 Printf
Helper classes, 55	
operator delete[]	murasaki::Debugger, 87
Helper classes, 55	Put 57 Oct. 10 Oct.
·	murasaki::FifoStrategy, 97
operator new	putMessage
Helper classes, 56	murasaki::LoggerStrategy, 117
operator new[]	murasaki::UartLogger, 149
Helper classes, 56	RePrint
PHY_AUTONEGO_COMPLETE	murasaki::Debugger, 88
stm32f7xx_hal_conf.h, 163	Receive
PHY AUTONEGOTIATION	murasaki::DebuggerUart, 93
stm32f7xx_hal_conf.h, 163	
	muracaki::I2CMactorStratogy 106
	murasaki::I2CMasterStrategy, 106
PHY_BCR	murasaki::l2cMaster, 101
PHY_BCR stm32f7xx_hal_conf.h, 163	murasaki::l2cMaster, 101 murasaki::l2cSlave, 110
PHY_BCR stm32f7xx_hal_conf.h, 163 PHY_BSR	murasaki::l2cMaster, 101 murasaki::l2cSlave, 110 murasaki::l2cSlaveStrategy, 114
PHY_BCR stm32f7xx_hal_conf.h, 163 PHY_BSR stm32f7xx_hal_conf.h, 163	murasaki::l2cMaster, 101 murasaki::l2cSlave, 110 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 144
PHY_BCR stm32f7xx_hal_conf.h, 163 PHY_BSR stm32f7xx_hal_conf.h, 163 PHY_DUPLEX_STATUS	murasaki::l2cMaster, 101 murasaki::l2cSlave, 110 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 144 murasaki::UartStrategy, 151
PHY_BCR stm32f7xx_hal_conf.h, 163 PHY_BSR stm32f7xx_hal_conf.h, 163 PHY_DUPLEX_STATUS stm32f7xx_hal_conf.h, 163	murasaki::l2cMaster, 101 murasaki::l2cSlave, 110 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 144 murasaki::UartStrategy, 151 ReceiveCompleteCallback
PHY_BCR stm32f7xx_hal_conf.h, 163 PHY_BSR stm32f7xx_hal_conf.h, 163 PHY_DUPLEX_STATUS stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_100M	murasaki::l2cMaster, 101 murasaki::l2cSlave, 110 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 144 murasaki::UartStrategy, 151 ReceiveCompleteCallback murasaki::DebuggerUart, 94
PHY_BCR stm32f7xx_hal_conf.h, 163 PHY_BSR stm32f7xx_hal_conf.h, 163 PHY_DUPLEX_STATUS stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_100M stm32f7xx_hal_conf.h, 163	murasaki::l2cMaster, 101 murasaki::l2cSlave, 110 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 144 murasaki::UartStrategy, 151 ReceiveCompleteCallback murasaki::DebuggerUart, 94 murasaki::l2CMasterStrategy, 106
PHY_BCR stm32f7xx_hal_conf.h, 163 PHY_BSR stm32f7xx_hal_conf.h, 163 PHY_DUPLEX_STATUS stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_100M stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_10M	murasaki::l2cMaster, 101 murasaki::l2cSlave, 110 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 144 murasaki::UartStrategy, 151 ReceiveCompleteCallback murasaki::DebuggerUart, 94 murasaki::l2cMasterStrategy, 106 murasaki::l2cMaster, 102
PHY_BCR stm32f7xx_hal_conf.h, 163 PHY_BSR stm32f7xx_hal_conf.h, 163 PHY_DUPLEX_STATUS stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_100M stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_10M stm32f7xx_hal_conf.h, 164	murasaki::l2cMaster, 101 murasaki::l2cSlave, 110 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 144 murasaki::UartStrategy, 151 ReceiveCompleteCallback murasaki::DebuggerUart, 94 murasaki::l2cMasterStrategy, 106 murasaki::l2cMaster, 102 murasaki::l2cSlave, 111
PHY_BCR stm32f7xx_hal_conf.h, 163 PHY_BSR stm32f7xx_hal_conf.h, 163 PHY_DUPLEX_STATUS stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_100M stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_100M	murasaki::l2cMaster, 101 murasaki::l2cSlave, 110 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 144 murasaki::UartStrategy, 151 ReceiveCompleteCallback murasaki::DebuggerUart, 94 murasaki::l2cMasterStrategy, 106 murasaki::l2cMaster, 102 murasaki::l2cSlave, 111 murasaki::l2cSlaveStrategy, 114
PHY_BCR stm32f7xx_hal_conf.h, 163 PHY_BSR stm32f7xx_hal_conf.h, 163 PHY_DUPLEX_STATUS stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_100M stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_100M stm32f7xx_hal_conf.h, 164	murasaki::l2cMaster, 101 murasaki::l2cSlave, 110 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 144 murasaki::UartStrategy, 151 ReceiveCompleteCallback murasaki::DebuggerUart, 94 murasaki::l2cMasterStrategy, 106 murasaki::l2cMaster, 102 murasaki::l2cSlave, 111 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 145
PHY_BCR stm32f7xx_hal_conf.h, 163 PHY_BSR stm32f7xx_hal_conf.h, 163 PHY_DUPLEX_STATUS stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_100M stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_100M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_10M	murasaki::l2cMaster, 101 murasaki::l2cSlave, 110 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 144 murasaki::UartStrategy, 151 ReceiveCompleteCallback murasaki::DebuggerUart, 94 murasaki::l2cMasterStrategy, 106 murasaki::l2cMaster, 102 murasaki::l2cSlave, 111 murasaki::l2cSlaveStrategy, 114
PHY_BCR stm32f7xx_hal_conf.h, 163 PHY_BSR stm32f7xx_hal_conf.h, 163 PHY_DUPLEX_STATUS stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_100M stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_100M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_10M stm32f7xx_hal_conf.h, 164	murasaki::l2cMaster, 101 murasaki::l2cSlave, 110 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 144 murasaki::UartStrategy, 151 ReceiveCompleteCallback murasaki::DebuggerUart, 94 murasaki::l2cMasterStrategy, 106 murasaki::l2cMaster, 102 murasaki::l2cSlave, 111 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 145
PHY_BCR stm32f7xx_hal_conf.h, 163 PHY_BSR stm32f7xx_hal_conf.h, 163 PHY_DUPLEX_STATUS stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_100M stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_100M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_ISOLATE	murasaki::l2cMaster, 101 murasaki::l2cSlave, 110 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 144 murasaki::UartStrategy, 151 ReceiveCompleteCallback murasaki::DebuggerUart, 94 murasaki::l2cMasterStrategy, 106 murasaki::l2cMaster, 102 murasaki::l2cSlave, 111 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 145 murasaki::Uart, 145 murasaki::UartStrategy, 151
PHY_BCR stm32f7xx_hal_conf.h, 163 PHY_BSR stm32f7xx_hal_conf.h, 163 PHY_DUPLEX_STATUS stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_100M stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_100M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_ISOLATE stm32f7xx_hal_conf.h, 164	murasaki::l2cMaster, 101 murasaki::l2cSlave, 110 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 144 murasaki::UartStrategy, 151 ReceiveCompleteCallback murasaki::DebuggerUart, 94 murasaki::l2cMasterStrategy, 106 murasaki::l2cMaster, 102 murasaki::l2cSlave, 111 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 145 murasaki::UartStrategy, 151 Release murasaki::Synchronizer, 136 RemoveSyslogFacilityFromMask
PHY_BCR stm32f7xx_hal_conf.h, 163 PHY_BSR stm32f7xx_hal_conf.h, 163 PHY_DUPLEX_STATUS stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_100M stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_100M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_ISOLATE stm32f7xx_hal_conf.h, 164 PHY_JABBER_DETECTION	murasaki::l2cMaster, 101 murasaki::l2cSlave, 110 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 144 murasaki::UartStrategy, 151 ReceiveCompleteCallback murasaki::DebuggerUart, 94 murasaki::l2cMasterStrategy, 106 murasaki::l2cMaster, 102 murasaki::l2cSlave, 111 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 145 murasaki::UartStrategy, 151 Release murasaki::Synchronizer, 136
PHY_BCR stm32f7xx_hal_conf.h, 163 PHY_BSR stm32f7xx_hal_conf.h, 163 PHY_DUPLEX_STATUS stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_100M stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_100M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_ISOLATE stm32f7xx_hal_conf.h, 164 PHY_JABBER_DETECTION stm32f7xx_hal_conf.h, 164	murasaki::I2cMaster, 101 murasaki::I2cSlave, 110 murasaki::I2cSlaveStrategy, 114 murasaki::Uart, 144 murasaki::UartStrategy, 151 ReceiveCompleteCallback murasaki::DebuggerUart, 94 murasaki::I2cMasterStrategy, 106 murasaki::I2cMaster, 102 murasaki::I2cSlave, 111 murasaki::I2cSlaveStrategy, 114 murasaki::Uart, 145 murasaki::UartStrategy, 151 Release murasaki::Synchronizer, 136 RemoveSyslogFacilityFromMask murasaki, 69
PHY_BCR stm32f7xx_hal_conf.h, 163 PHY_BSR stm32f7xx_hal_conf.h, 163 PHY_DUPLEX_STATUS stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_100M stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_100M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_ISOLATE stm32f7xx_hal_conf.h, 164 PHY_JABBER_DETECTION stm32f7xx_hal_conf.h, 164 PHY_LINKED_STATUS	murasaki::l2cSlave, 110 murasaki::l2cSlave, 110 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 144 murasaki::UartStrategy, 151 ReceiveCompleteCallback murasaki::DebuggerUart, 94 murasaki::l2cMasterStrategy, 106 murasaki::l2cMaster, 102 murasaki::l2cSlave, 111 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 145 murasaki::UartStrategy, 151 Release murasaki::Synchronizer, 136 RemoveSyslogFacilityFromMask murasaki, 69 STM32F7xx_System_Private_Defines, 61
PHY_BCR stm32f7xx_hal_conf.h, 163 PHY_BSR stm32f7xx_hal_conf.h, 163 PHY_DUPLEX_STATUS stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_100M stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_100M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_JABBER_DETECTION stm32f7xx_hal_conf.h, 164 PHY_LINKED_STATUS stm32f7xx_hal_conf.h, 164	murasaki::l2cSlave, 110 murasaki::l2cSlave, 110 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 144 murasaki::UartStrategy, 151 ReceiveCompleteCallback murasaki::DebuggerUart, 94 murasaki::l2cMasterStrategy, 106 murasaki::l2cMaster, 102 murasaki::l2cSlave, 111 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 145 murasaki::UartStrategy, 151 Release murasaki::Synchronizer, 136 RemoveSyslogFacilityFromMask murasaki, 69  STM32F7xx_System_Private_Defines, 61 VECT_TAB_OFFSET, 61
PHY_BCR stm32f7xx_hal_conf.h, 163 PHY_BSR stm32f7xx_hal_conf.h, 163 PHY_DUPLEX_STATUS stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_100M stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_100M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_ISOLATE stm32f7xx_hal_conf.h, 164 PHY_JABBER_DETECTION stm32f7xx_hal_conf.h, 164 PHY_LINKED_STATUS stm32f7xx_hal_conf.h, 164 PHY_LINKED_STATUS stm32f7xx_hal_conf.h, 164 PHY_LOOPBACK	murasaki::l2cSlave, 110 murasaki::l2cSlaveStrategy, 114 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 144 murasaki::UartStrategy, 151 ReceiveCompleteCallback murasaki::DebuggerUart, 94 murasaki::l2cMasterStrategy, 106 murasaki::l2cMaster, 102 murasaki::l2cSlave, 111 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 145 murasaki::UartStrategy, 151 Release murasaki::Synchronizer, 136 RemoveSyslogFacilityFromMask murasaki, 69  STM32F7xx_System_Private_Defines, 61 VECT_TAB_OFFSET, 61 STM32F7xx_System_Private_FunctionPrototypes, 64
PHY_BCR stm32f7xx_hal_conf.h, 163 PHY_BSR stm32f7xx_hal_conf.h, 163 PHY_DUPLEX_STATUS stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_100M stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_100M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_ISOLATE stm32f7xx_hal_conf.h, 164 PHY_JABBER_DETECTION stm32f7xx_hal_conf.h, 164 PHY_LINKED_STATUS stm32f7xx_hal_conf.h, 164 PHY_LOOPBACK stm32f7xx_hal_conf.h, 164	murasaki::l2cMaster, 101 murasaki::l2cSlave, 110 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 144 murasaki::UartStrategy, 151 ReceiveCompleteCallback murasaki::DebuggerUart, 94 murasaki::l2cMasterStrategy, 106 murasaki::l2cMaster, 102 murasaki::l2cSlave, 111 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 145 murasaki::UartStrategy, 151 Release murasaki::Synchronizer, 136 RemoveSyslogFacilityFromMask murasaki, 69  STM32F7xx_System_Private_Defines, 61 VECT_TAB_OFFSET, 61 STM32F7xx_System_Private_FunctionPrototypes, 64 STM32F7xx_System_Private_Functions, 65
PHY_BCR stm32f7xx_hal_conf.h, 163 PHY_BSR stm32f7xx_hal_conf.h, 163 PHY_DUPLEX_STATUS stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_100M stm32f7xx_hal_conf.h, 163 PHY_FULLDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_100M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_HALFDUPLEX_10M stm32f7xx_hal_conf.h, 164 PHY_ISOLATE stm32f7xx_hal_conf.h, 164 PHY_JABBER_DETECTION stm32f7xx_hal_conf.h, 164 PHY_LINKED_STATUS stm32f7xx_hal_conf.h, 164 PHY_LINKED_STATUS stm32f7xx_hal_conf.h, 164 PHY_LOOPBACK	murasaki::l2cSlave, 110 murasaki::l2cSlaveStrategy, 114 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 144 murasaki::UartStrategy, 151 ReceiveCompleteCallback murasaki::DebuggerUart, 94 murasaki::l2cMasterStrategy, 106 murasaki::l2cMaster, 102 murasaki::l2cSlave, 111 murasaki::l2cSlaveStrategy, 114 murasaki::Uart, 145 murasaki::UartStrategy, 151 Release murasaki::Synchronizer, 136 RemoveSyslogFacilityFromMask murasaki, 69  STM32F7xx_System_Private_Defines, 61 VECT_TAB_OFFSET, 61 STM32F7xx_System_Private_FunctionPrototypes, 64

STM32F7xx System Private Includes, 59	murasaki::TaskStrategy, 141
HSE VALUE, 59	start
HSI VALUE, 59	murasaki::Adau1361, 75
STM32F7xx_System_Private_Macros, 62	murasaki::AudioCodecStrategy, 77
STM32F7xx System Private TypesDefinitions, 60	StartDefaultTask
STM32F7xx_System_Private_Variables, 63	main.c, 227
	stm32f7xx_hal_conf.h
send_command	assert_failed, 165
murasaki::Adau1361, 73 send_command_table	assert_param, 162
murasaki::Adau1361, 73	EXTERNAL_CLOCK_VALUE, 162
Set	HSE STARTUP TIMEOUT, 162
murasaki::BitOut, 83	HSE_VALUE, 162
murasaki::BitOutStrategy, 85	HSI_VALUE, 162
set_aux_input_gain	LSE_STARTUP_TIMEOUT, 163
murasaki::Adau1361, 73	LSE VALUE, 163
murasaki::AudioCodecStrategy, 76	LSI VALUE, 163
set_hp_output_gain	PHY_AUTONEGO_COMPLETE, 163
murasaki::Adau1361, 74	PHY AUTONEGOTIATION, 163
murasaki::AudioCodecStrategy, 76	PHY BCR, 163
set_line_input_gain	PHY BSR, 163
murasaki::Adau1361, 74	PHY DUPLEX STATUS, 163
murasaki::AudioCodecStrategy, 77	PHY_FULLDUPLEX_100M, 163
set_line_output_gain	PHY_FULLDUPLEX_10M, 164
murasaki::Adau1361, 74	PHY HALFDUPLEX 100M, 164
murasaki::AudioCodecStrategy, 77	PHY HALFDUPLEX 10M, 164
	PHY ISOLATE, 164
set_mic_input_gain	PHY_JABBER_DETECTION, 164
murasaki::AudioCodecStrategy, 77 SetHardwareFlowControl	PHY_LINKED_STATUS, 164
	PHY LOOPBACK, 164
murasaki::DebuggerUart, 94	PHY POWERDOWN, 164
murasaki::Uart, 145	PHY_RESET, 164
murasaki::UartStrategy, 152 SetPostMortem	PHY_RESTART_AUTONEGOTIATION, 164
	PHY_SPEED_STATUS, 165
murasaki::DebuggerFifo, 90	PHY SR, 165
SetSpeed	TICK INT PRIORITY, 165
murasaki::DebuggerUart, 94	VDD_VALUE, 165
murasaki::Uart, 145	stm32f7xx_it.c
murasaki::UartStrategy, 152	hdma_spi1_tx, 232
SetSyslogFacilityMask	Stm32f7xx_system, 58
murasaki, 69	Synchronization and Exclusive access, 39
SetSyslogSererityThreshold	SyslogFacility
murasaki, 69	Definitions and Configuration, 43
severity_	SyslogSeverity
murasaki::Debugger, 88	Definitions and Configuration, 44
SpiClockPhase	SystemClock_Config
Definitions and Configuration, 42	main.c, 228
SpiClockPolarity	SystemCoreClockUpdate
Definitions and Configuration, 43	STM32F7xx_System_Private_Functions, 65
SpiMaster	SystemInit
murasaki::SpiMaster, 121	STM32F7xx_System_Private_Functions, 66
SpiSlave	
murasaki::SpiSlave, 127	TICK_INT_PRIORITY
SpiSlaveSpecifier	stm32f7xx_hal_conf.h, 165
murasaki::SpiSlaveSpecifier, 130	Task
SpiSlaveSpecifierStrategy	murasaki::Task, 138
murasaki::SpiSlaveSpecifierStrategy, 132, 133	TaskBody
SpiStatus	murasaki::Task, 139
Definitions and Configuration, 43	murasaki::TaskStrategy, 141
Start	TaskStrategy

```
murasaki::TaskStrategy, 140
Third party classes, 40
Transmit
     murasaki::DebuggerUart, 95
     murasaki::I2CMasterStrategy, 106
     murasaki::I2cMaster, 102
     murasaki::I2cSlave, 111
     murasaki::12cSlaveStrategy, 115
     murasaki::Uart, 146
     murasaki::UartStrategy, 152
TransmitAndReceive
    murasaki::SpiMaster, 122
     murasaki::SpiMasterStrategy, 125
     murasaki::SpiSlave, 128
     murasaki::SpiSlaveStrategy, 135
TransmitAndReceiveCompleteCallback
     murasaki::SpiMaster, 123
     murasaki::SpiMasterStrategy, 125
     murasaki::SpiSlave, 129
     murasaki::SpiSlaveStrategy, 136
TransmitCompleteCallback
     murasaki::DebuggerUart, 95
     murasaki::I2CMasterStrategy, 107
    murasaki::I2cMaster, 103
     murasaki::12cSlave, 112
     murasaki::I2cSlaveStrategy, 115
     murasaki::Uart, 146
    murasaki::UartStrategy, 153
TransmitThenReceive
     murasaki::I2CMasterStrategy, 107
     murasaki::I2cMaster, 103
Uart
     murasaki::Uart, 143
UartHardwareFlowControl
     Definitions and Configuration, 44
UartLogger
     murasaki::UartLogger, 148
UartStatus
     Definitions and Configuration, 45
UartTimeout
     Definitions and Configuration, 45
VDD_VALUE
     stm32f7xx_hal_conf.h, 165
VECT TAB OFFSET
     STM32F7xx_System_Private_Defines, 61
     murasaki::Synchronizer, 136
wait_pll_lock
     murasaki::Adau1361, 75
WaitMilliSeconds
     Definitions and Configuration, 45
```