

Autonomous Underwater Submarines

Generated by Doxygen 1.8.8

Thu Apr 30 2015 11:03:55

Contents

- 1 Brief Description** **1**

- 2 File Index** **3**
 - 2.1 File List 3

- 3 File Documentation** **7**
 - 3.1 Algorithms/Motor_Control/Motor_Control.c File Reference 7
 - 3.1.1 Detailed Description 8
 - 3.1.2 Macro Definition Documentation 8
 - 3.1.2.1 kdZ 8
 - 3.1.2.2 kiZ 9
 - 3.1.2.3 kpZ 9
 - 3.1.2.4 Motor_Z_Control_Type 9
 - 3.1.2.5 motorZ_const 9
 - 3.1.2.6 Velocity_Time_Difference 9
 - 3.1.3 Function Documentation 9
 - 3.1.3.1 Motor_Control 9
 - 3.1.3.2 Motor_Control_Enable_Flag 10
 - 3.1.3.3 Motor_Control_Motor_X_Flag_Update 11
 - 3.1.3.4 Motor_Control_Motor_Y_Flag_Update 12
 - 3.1.3.5 Motor_Control_Motor_Z_Flag_Update 13
 - 3.1.3.6 Motor_Control_Update 14
 - 3.1.3.7 Motor_Control_Update_Depth_Z 15
 - 3.1.3.8 Motor_Control_Update_Velocity_X 15
 - 3.1.3.9 Motor_Control_Update_Velocity_Y 16
 - 3.1.3.10 Motor_Control_X_Open 16
 - 3.1.3.11 Motor_Control_Y_Open 17
 - 3.1.3.12 Motor_Control_Z 18
 - 3.2 Motor_Control.c 19
 - 3.3 Algorithms/Motor_Control/Motor_Control.h File Reference 22
 - 3.3.1 Detailed Description 23
 - 3.3.2 Function Documentation 23

3.3.2.1	Motor_Control	23
3.3.2.2	Motor_Control_Enable_Flag	24
3.3.2.3	Motor_Control_Motor_X_Flag_Update	25
3.3.2.4	Motor_Control_Motor_Y_Flag_Update	26
3.3.2.5	Motor_Control_Motor_Z_Flag_Update	27
3.3.2.6	Motor_Control_Update	28
3.3.2.7	Motor_Control_Update_Depth_Z	29
3.3.2.8	Motor_Control_Update_Velocity_X	29
3.3.2.9	Motor_Control_Update_Velocity_Y	30
3.4	Motor_Control.h	30
3.5	Algorithms/Navigation/Swarming.c File Reference	31
3.5.1	Detailed Description	32
3.5.2	Macro Definition Documentation	32
3.5.2.1	Photodiode_Ratio_Maximum	32
3.5.2.2	Photodiode_Ratio_Minimum	32
3.5.3	Enumeration Type Documentation	32
3.5.3.1	Region_Value	32
3.5.4	Function Documentation	33
3.5.4.1	Swarming	33
3.5.4.2	Swarming_Enable_Flag	34
3.5.4.3	Swarming_Forward_Calculation	34
3.5.4.4	Swarming_Left_Calculation	35
3.5.4.5	Swarming_Range_Left_Right	35
3.5.4.6	Swarming_Region	36
3.5.4.7	Swarming_Retrieve_X_Offset	36
3.5.4.8	Swarming_Retrieve_Y_Offset	36
3.5.4.9	Swarming_Right_Calculation	37
3.5.4.10	Swarming_Update	37
3.6	Swarming.c	38
3.7	Algorithms/Navigation/Swarming.h File Reference	43
3.7.1	Detailed Description	43
3.7.2	Enumeration Type Documentation	44
3.7.2.1	Region_Radius	44
3.7.3	Function Documentation	44
3.7.3.1	Swarming	44
3.7.3.2	Swarming_Enable_Flag	45
3.7.3.3	Swarming_Retrieve_X_Offset	46
3.7.3.4	Swarming_Retrieve_Y_Offset	46
3.7.3.5	Swarming_Update	46
3.8	Swarming.h	47

3.9 Algorithms/State_Machine/State_Machine.c File Reference	48
3.9.1 Detailed Description	49
3.9.2 Function Documentation	49
3.9.2.1 State_Machine_Enable_Flag	49
3.9.2.2 State_Machine_Initialization	49
3.9.2.3 State_Machine_State_1	50
3.9.2.4 State_Machine_State_2	51
3.9.2.5 State_Machine_State_3	52
3.9.2.6 State_Machine_State_4	52
3.9.2.7 State_Machine_State_5	53
3.9.2.8 State_Machine_State_6	54
3.9.2.9 State_Machine_Trasition_2	55
3.9.2.10 State_Machine_Trasition_3	56
3.9.2.11 State_Machine_Trasition_4	57
3.9.2.12 State_Machine_Trasition_5	58
3.9.2.13 State_Machine_Trasition_6	59
3.9.2.14 State_Machine_Update	60
3.10 State_Machine.c	62
3.11 Algorithms/State_Machine/State_Machine.h File Reference	66
3.11.1 Detailed Description	67
3.11.2 Function Documentation	67
3.11.2.1 State_Machine_Enable_Flag	67
3.11.2.2 State_Machine_Update	67
3.12 State_Machine.h	69
3.13 Main.c File Reference	69
3.13.1 Detailed Description	69
3.13.2 Function Documentation	70
3.13.2.1 main	70
3.14 Main.c	71
3.15 Main_Program.c File Reference	72
3.15.1 Detailed Description	73
3.15.2 Function Documentation	73
3.15.2.1 Main_Program	73
3.16 Main_Program.c	74
3.17 Main_Program.h File Reference	75
3.17.1 Detailed Description	75
3.17.2 Function Documentation	76
3.17.2.1 Main_Program	76
3.18 Main_Program.h	77
3.19 mainpage.dox File Reference	77

3.20	Master_Build_Control.h File Reference	77
3.20.1	Detailed Description	79
3.20.2	Macro Definition Documentation	79
3.20.2.1	Compass_Test_Mode	79
3.20.2.2	H_Bridge_Print	79
3.20.2.3	H_Bridge_Test_Direction	79
3.20.2.4	H_Bridge_Test_Mode	79
3.20.2.5	LED_Test_Mode	80
3.20.2.6	Main_Mode	80
3.20.2.7	Motor_Control_Test_Mode	80
3.20.2.8	Multiplexer_Test_Mode	80
3.20.2.9	Photodiode_Test_Mode	80
3.20.2.10	Print_Motor_Information	81
3.20.2.11	Print_Region	81
3.20.2.12	Print_State	81
3.20.2.13	State_Machine_Test_Mode	81
3.21	Master_Build_Control.h	81
3.22	Microcontroller/ADC/ADC.c File Reference	82
3.22.1	Detailed Description	83
3.22.2	Macro Definition Documentation	84
3.22.2.1	VREF	84
3.22.3	Function Documentation	84
3.22.3.1	ADC_Convert_To_Voltage	84
3.22.3.2	ADC_Read_Pin	84
3.22.3.3	ADC_Setup_Pin	85
3.23	ADC.c	87
3.24	Microcontroller/ADC/ADC.h File Reference	88
3.24.1	Detailed Description	89
3.24.2	Function Documentation	90
3.24.2.1	ADC_Convert_To_Voltage	90
3.24.2.2	ADC_Read_Pin	91
3.24.2.3	ADC_Setup_Pin	92
3.25	ADC.h	94
3.26	Microcontroller/ADC/CD4051.c File Reference	95
3.26.1	Detailed Description	96
3.26.2	Macro Definition Documentation	96
3.26.2.1	ADC_DEFAULT_PIN	96
3.26.2.2	F_CPU	96
3.26.3	Function Documentation	96
3.26.3.1	CD4051_Initialize	96

3.26.3.2	CD4051_Read_Range	98
3.26.3.3	CD4051_Read_Single_Pin	99
3.27	CD4051.c	100
3.28	Microcontroller/ADC/CD4051.h File Reference	101
3.28.1	Detailed Description	102
3.28.2	Macro Definition Documentation	103
3.28.2.1	Battery_Mux_Port	103
3.28.2.2	Photodiode_front	103
3.28.2.3	Photodiode_leftfront	103
3.28.2.4	Photodiode_leftrear	103
3.28.2.5	Photodiode_rightfront	103
3.28.2.6	Photodiode_rightrear	103
3.28.2.7	Pressure_Sensor_Mux_Port	103
3.28.3	Function Documentation	104
3.28.3.1	CD4051_Initialize	104
3.28.3.2	CD4051_Read_Range	105
3.28.3.3	CD4051_Read_Single_Pin	106
3.29	CD4051.h	107
3.30	Microcontroller/Communication/I2C_Interface.c File Reference	108
3.30.1	Detailed Description	109
3.30.2	Macro Definition Documentation	109
3.30.2.1	F_CPU	109
3.30.3	Function Documentation	109
3.30.3.1	I2C_Interface_Initialize	109
3.30.3.2	I2C_Interface_Not_Busy	111
3.30.3.3	I2C_Interface_Read_Array	112
3.30.3.4	I2C_Interface_Single_Read	113
3.30.3.5	I2C_Interface_Write	114
3.31	I2C_Interface.c	115
3.32	Microcontroller/Communication/I2C_Interface.h File Reference	116
3.32.1	Detailed Description	117
3.32.2	Function Documentation	118
3.32.2.1	I2C_Interface_Initialize	118
3.32.2.2	I2C_Interface_Not_Busy	119
3.32.2.3	I2C_Interface_Read_Array	120
3.32.2.4	I2C_Interface_Single_Read	121
3.32.2.5	I2C_Interface_Write	122
3.33	I2C_Interface.h	123
3.34	Microcontroller/Communication/TWI_Master.c File Reference	123
3.34.1	Detailed Description	124

3.34.2	Function Documentation	125
3.34.2.1	ISR	125
3.34.2.2	TWI_Get_Data_From_Transceiver	125
3.34.2.3	TWI_Get_State_Info	126
3.34.2.4	TWI_Master_Initialise	126
3.34.2.5	TWI_Start_Transceiver	128
3.34.2.6	TWI_Start_Transceiver_With_Data	128
3.34.2.7	TWI_Transceiver_Busy	129
3.35	TWI_Master.c	129
3.36	Microcontroller/Communication/TWI_Master.h File Reference	132
3.36.1	Detailed Description	135
3.36.2	Macro Definition Documentation	135
3.36.2.1	F_CPU	135
3.36.2.2	FALSE	136
3.36.2.3	MESSAGE	136
3.36.2.4	TRUE	136
3.36.2.5	TWI_ADR_BITS	136
3.36.2.6	TWI_ARB_LOST	136
3.36.2.7	TWI_BITRATE	136
3.36.2.8	TWI_BUFFER_SIZE	136
3.36.2.9	TWI_BUS_ERROR	136
3.36.2.10	TWI_MRX_ADR_ACK	136
3.36.2.11	TWI_MRX_ADR_NACK	137
3.36.2.12	TWI_MRX_DATA_ACK	137
3.36.2.13	TWI_MRX_DATA_NACK	137
3.36.2.14	TWI_MTX_ADR_ACK	137
3.36.2.15	TWI_MTX_ADR_NACK	137
3.36.2.16	TWI_MTX_DATA_ACK	137
3.36.2.17	TWI_MTX_DATA_NACK	137
3.36.2.18	TWI_NO_STATE	138
3.36.2.19	TWI_PSBP	138
3.36.2.20	TWI_READ_BIT	138
3.36.2.21	TWI_REP_START	138
3.36.2.22	TWI_SRX_ADR_ACK	138
3.36.2.23	TWI_SRX_ADR_ACK_M_ARB_LOST	138
3.36.2.24	TWI_SRX_ADR_DATA_ACK	138
3.36.2.25	TWI_SRX_ADR_DATA_NACK	138
3.36.2.26	TWI_SRX_GEN_ACK	138
3.36.2.27	TWI_SRX_GEN_ACK_M_ARB_LOST	139
3.36.2.28	TWI_SRX_GEN_DATA_ACK	139

3.36.2.29	TWI_SRX_GEN_DATA_NACK	139
3.36.2.30	TWI_SRX_STOP_RESTART	139
3.36.2.31	TWI_START	139
3.36.2.32	TWI_STX_ADR_ACK	139
3.36.2.33	TWI_STX_ADR_ACK_M_ARB_LOST	139
3.36.2.34	TWI_STX_DATA_ACK	139
3.36.2.35	TWI_STX_DATA_ACK_LAST_BYTE	139
3.36.2.36	TWI_STX_DATA_NACK	140
3.36.2.37	TWI_TWBR	140
3.36.2.38	TWI_TWPS	140
3.36.3	Function Documentation	140
3.36.3.1	TWI_Get_Data_From_Transceiver	140
3.36.3.2	TWI_Get_State_Info	141
3.36.3.3	TWI_Master_Initialise	141
3.36.3.4	TWI_Start_Transceiver	143
3.36.3.5	TWI_Start_Transceiver_With_Data	143
3.36.3.6	TWI_Transceiver_Busy	144
3.37	TWI_Master.h	144
3.38	Microcontroller/Communication/USART.c File Reference	146
3.38.1	Detailed Description	147
3.38.2	Macro Definition Documentation	147
3.38.2.1	BAUD	147
3.38.2.2	FOSC	147
3.38.2.3	MYUBRR	148
3.38.2.4	PRECISION	148
3.38.3	Function Documentation	148
3.38.3.1	USART_ftoa	148
3.38.3.2	USART_Initialize	149
3.38.3.3	USART_Read_String	151
3.38.3.4	USART_Read_String_With_Echo	152
3.38.3.5	USART_Receive_Byte	152
3.38.3.6	USART_Send_Byte	153
3.38.3.7	USART_Send_String	154
3.39	USART.c	156
3.40	Microcontroller/Communication/USART.h File Reference	159
3.40.1	Detailed Description	159
3.40.2	Macro Definition Documentation	160
3.40.2.1	FLOATING_POINT_BUFFER_SIZE	160
3.40.3	Function Documentation	160
3.40.3.1	USART_ftoa	160

3.40.3.2	USART_Initialize	161
3.40.3.3	USART_Read_String	163
3.40.3.4	USART_Read_String_With_Echo	164
3.40.3.5	USART_Receive_Byte	164
3.40.3.6	USART_Send_Byte	165
3.40.3.7	USART_Send_String	166
3.41	USART.h	168
3.42	Microcontroller/Interrupt/Interrupt.c File Reference	169
3.42.1	Detailed Description	169
3.42.2	Function Documentation	170
3.42.2.1	ISR	170
3.42.2.2	ISR	170
3.42.2.3	ISR	171
3.43	Interrupt.c	171
3.44	Microcontroller/Interrupt/Interrupt.h File Reference	172
3.45	Interrupt.h	172
3.46	Microcontroller/Interrupt/Timer.c File Reference	172
3.46.1	Detailed Description	173
3.46.2	Function Documentation	174
3.46.2.1	Timer0_Disable	174
3.46.2.2	Timer0_Enable	174
3.46.2.3	Timer0_Initialize	174
3.46.2.4	Timer1_Disable	175
3.46.2.5	Timer1_Enable	175
3.46.2.6	Timer1_Initialize	176
3.46.2.7	Timer2_Disable	176
3.46.2.8	Timer2_Enable	176
3.46.2.9	Timer2_Initialize	176
3.46.2.10	Timer_Initialize	177
3.47	Timer.c	177
3.48	Microcontroller/Interrupt/Timer.h File Reference	179
3.48.1	Detailed Description	179
3.48.2	Function Documentation	180
3.48.2.1	Timer0_Disable	180
3.48.2.2	Timer0_Enable	180
3.48.2.3	Timer1_Disable	181
3.48.2.4	Timer1_Enable	181
3.48.2.5	Timer2_Disable	181
3.48.2.6	Timer2_Enable	181
3.48.2.7	Timer_Initialize	182

3.49	Timer.h	182
3.50	Subsystem/H_Bridge/DRV8830.c File Reference	183
3.50.1	Detailed Description	185
3.50.2	Macro Definition Documentation	185
3.50.2.1	Clear_Fault_Bit	185
3.50.2.2	H_Bridge_Max_Voltage_VSET	185
3.50.2.3	H_Bridge_Min_Voltage	185
3.50.2.4	H_Bridge_Min_Voltage_VSET	186
3.50.2.5	H_Bridge_Voltage_Conversion	186
3.50.2.6	Motor_Fault_Count_Max	186
3.50.2.7	Motor_Max_Jump	186
3.50.3	Enumeration Type Documentation	186
3.50.3.1	DRV8830_Modes	186
3.50.3.2	DRV8830_Registers	186
3.50.3.3	Motor_Adress	187
3.50.4	Function Documentation	187
3.50.4.1	DRV8830_Check_Motor_X	187
3.50.4.2	DRV8830_Check_Motor_Y	187
3.50.4.3	DRV8830_Check_Motor_Z	188
3.50.4.4	DRV8830_Check_Motors	189
3.50.4.5	DRV8830_Clear_Fault	190
3.50.4.6	DRV8830_Clear_Fault_Motor_X	191
3.50.4.7	DRV8830_Clear_Fault_Motor_Y	192
3.50.4.8	DRV8830_Clear_Fault_Motor_Z	192
3.50.4.9	DRV8830_Initialize	193
3.50.4.10	DRV8830_Jump_Check	194
3.50.4.11	DRV8830_Mode	196
3.50.4.12	DRV8830_Read_Fault	197
3.50.4.13	DRV8830_Read_Fault_Motor_X	199
3.50.4.14	DRV8830_Read_Fault_Motor_Y	199
3.50.4.15	DRV8830_Read_Fault_Motor_Z	200
3.50.4.16	DRV8830_Reset_Motor_X	201
3.50.4.17	DRV8830_Reset_Motor_Y	202
3.50.4.18	DRV8830_Reset_Motor_Z	202
3.50.4.19	DRV8830_Set_Voltage	203
3.50.4.20	DRV8830_Set_Voltage_Motor_X	205
3.50.4.21	DRV8830_Set_Voltage_Motor_Y	207
3.50.4.22	DRV8830_Set_Voltage_Motor_Z	209
3.50.4.23	DRV8830_Start_Up_Flag_Control	211
3.50.4.24	DRV8830_Voltage_To_Int	213

3.51	DRV8830.c	215
3.52	Subsystem/H_Bridge/DRV8830.h File Reference	223
3.52.1	Detailed Description	224
3.52.2	Macro Definition Documentation	225
3.52.2.1	Motor_Max_Voltage	225
3.52.3	Function Documentation	225
3.52.3.1	DRV8830_Check_Motors	225
3.52.3.2	DRV8830_Clear_Fault_Motor_X	226
3.52.3.3	DRV8830_Clear_Fault_Motor_Y	227
3.52.3.4	DRV8830_Clear_Fault_Motor_Z	227
3.52.3.5	DRV8830_Initialize	228
3.52.3.6	DRV8830_Read_Fault_Motor_X	229
3.52.3.7	DRV8830_Read_Fault_Motor_Y	230
3.52.3.8	DRV8830_Read_Fault_Motor_Z	231
3.52.3.9	DRV8830_Set_Voltage_Motor_X	232
3.52.3.10	DRV8830_Set_Voltage_Motor_Y	233
3.52.3.11	DRV8830_Set_Voltage_Motor_Z	235
3.53	DRV8830.h	237
3.54	Subsystem/IMU/LSM303.c File Reference	237
3.54.1	Detailed Description	239
3.54.2	Macro Definition Documentation	239
3.54.2.1	Current_Device	239
3.54.2.2	D_SA0_HIGH_ADDRESS	239
3.54.2.3	D_SA0_LOW_ADDRESS	239
3.54.2.4	D_WHO_ID	239
3.54.2.5	Device_D	239
3.54.2.6	Device_DLH	239
3.54.2.7	Device_DLHC	239
3.54.2.8	Device_DLM	240
3.54.2.9	DLM_WHO_ID	240
3.54.2.10	Heading_Stable_Value	240
3.54.2.11	NON_D_ACC_SA0_HIGH_ADDRESS	240
3.54.2.12	NON_D_ACC_SA0_LOW_ADDRESS	240
3.54.2.13	NON_D_MAG_ADDRESS	240
3.54.2.14	Radians_To_Degrees_Conv	240
3.54.2.15	Register_Count	240
3.54.2.16	TEST_REG_INVALID	240
3.54.2.17	X	240
3.54.2.18	Y	241
3.54.2.19	Z	241

3.54.3	Enumeration Type Documentation	241
3.54.3.1	Register_Address	241
3.54.4	Function Documentation	242
3.54.4.1	LSM303_Array_Cross_Product	242
3.54.4.2	LSM303_Array_Dot_Product	242
3.54.4.3	LSM303_Array_Normalize	243
3.54.4.4	LSM303_Heading	243
3.54.4.5	LSM303_HeadingWithOffset	244
3.54.4.6	LSM303_Initialize	244
3.54.4.7	LSM303_IsStablized	245
3.54.4.8	LSM303_Read_Acc	245
3.54.4.9	LSM303_Read_Mag	246
3.54.4.10	LSM303_RotationFinished	246
3.54.4.11	LSM303_Test_Register	247
3.54.4.12	LSM303_UpdateForwardHeading	247
3.55	LSM303.c	248
3.56	Subsystem/IMU/LSM303.h File Reference	253
3.56.1	Detailed Description	254
3.56.2	Function Documentation	254
3.56.2.1	LSM303_Heading	254
3.56.2.2	LSM303_HeadingWithOffset	255
3.56.2.3	LSM303_Initialize	255
3.56.2.4	LSM303_IsStablized	256
3.56.2.5	LSM303_Read_Acc	256
3.56.2.6	LSM303_Read_Mag	257
3.56.2.7	LSM303_RotationFinished	257
3.56.2.8	LSM303_UpdateForwardHeading	258
3.57	LSM303.h	258
3.58	Subsystem/Light/CLS15.c File Reference	259
3.58.1	Detailed Description	260
3.58.2	Function Documentation	260
3.58.2.1	CLS15_Ratio_Calculation	260
3.58.2.2	CLS15_Retrieve_Phodiode_ADC_Front	261
3.58.2.3	CLS15_Retrieve_Phodiode_ADC_Left_Front	261
3.58.2.4	CLS15_Retrieve_Phodiode_ADC_Left_Rear	262
3.58.2.5	CLS15_Retrieve_Phodiode_ADC_Right_Front	262
3.58.2.6	CLS15_Retrieve_Phodiode_ADC_Right_Rear	263
3.58.2.7	CLS15_Retrieve_Phodiode_Ratio_Left	263
3.58.2.8	CLS15_Retrieve_Phodiode_Ratio_Right	264
3.58.2.9	CLS15_Update_Phodiode_Offset	264

3.58.2.10	CLS15_Update_Photodiode_Offset_Front	265
3.58.2.11	CLS15_Update_Photodiode_Offset_Left_Front	266
3.58.2.12	CLS15_Update_Photodiode_Offset_Left_Rear	266
3.58.2.13	CLS15_Update_Photodiode_Offset_Right_Front	266
3.58.2.14	CLS15_Update_Photodiode_Offset_Right_Rear	267
3.58.2.15	CLS15_Update_Photodiode_Value_Front	267
3.58.2.16	CLS15_Update_Photodiode_Value_Left	268
3.58.2.17	CLS15_Update_Photodiode_Value_Right	269
3.58.2.18	CLS15_Update_Photodiode_Values	270
3.59	CLS15.c	271
3.60	Subsystem/Light/CLS15.h File Reference	274
3.60.1	Detailed Description	274
3.60.2	Function Documentation	275
3.60.2.1	CLS15_Retrieve_Photodiode_ADC_Front	275
3.60.2.2	CLS15_Retrieve_Photodiode_ADC_Left_Front	275
3.60.2.3	CLS15_Retrieve_Photodiode_ADC_Left_Rear	276
3.60.2.4	CLS15_Retrieve_Photodiode_ADC_Rear	276
3.60.2.5	CLS15_Retrieve_Photodiode_ADC_Right_Front	277
3.60.2.6	CLS15_Retrieve_Photodiode_ADC_Right_Rear	277
3.60.2.7	CLS15_Retrieve_Photodiode_Ratio_Left	278
3.60.2.8	CLS15_Retrieve_Photodiode_Ratio_Right	278
3.60.2.9	CLS15_Update_Photodiode_Offset	279
3.60.2.10	CLS15_Update_Photodiode_Values	279
3.61	CLS15.h	280
3.62	Subsystem/Light/XPEBBL.c File Reference	281
3.62.1	Detailed Description	281
3.62.2	Macro Definition Documentation	282
3.62.2.1	Cycle_State	282
3.62.2.2	Front_LED_Bit	282
3.62.3	Function Documentation	282
3.62.3.1	XPEBBL_Initialize	282
3.62.3.2	XPEBBL_Set_Pin	283
3.62.3.3	XPEBBL_Toggle_Front_LED	284
3.62.3.4	XPEBLL_Change_LED	285
3.63	XPEBBL.c	285
3.64	Subsystem/Light/XPEBBL.h File Reference	287
3.64.1	Detailed Description	287
3.64.2	Function Documentation	288
3.64.2.1	XPEBBL_Initialize	288
3.64.2.2	XPEBBL_Set_Pin	288

3.64.2.3	XPEBBL_Toggle_Front_LED	289
3.64.2.4	XPEBLL_Change_LED	290
3.65	XPEBBL.h	290
3.66	Subsystem/Power/Battery.c File Reference	290
3.66.1	Detailed Description	291
3.66.2	Macro Definition Documentation	292
3.66.2.1	Battery_Safe_Voltage	292
3.66.3	Function Documentation	292
3.66.3.1	Battery_Check_Status	292
3.66.3.2	Battery_Read_ADC	292
3.66.3.3	Battery_Read_Voltage	293
3.67	Battery.c	294
3.68	Subsystem/Power/Battery.h File Reference	294
3.68.1	Detailed Description	295
3.68.2	Function Documentation	295
3.68.2.1	Battery_Check_Status	295
3.68.2.2	Battery_Read_Voltage	296
3.69	Battery.h	296
3.70	Subsystem/Pressure/MPX5010GP.c File Reference	296
3.70.1	Detailed Description	297
3.70.2	Macro Definition Documentation	298
3.70.2.1	Max_PSI	298
3.70.2.2	V_Offset	298
3.70.2.3	VFSS	298
3.70.3	Function Documentation	298
3.70.3.1	MPX5010GP_Calculate_PSI	298
3.70.3.2	MPX5010GP_Depth	299
3.70.3.3	MPX5010GP_Read_Value	300
3.71	MPX5010GP.c	301
3.72	Subsystem/Pressure/MPX5010GP.h File Reference	301
3.72.1	Detailed Description	302
3.72.2	Macro Definition Documentation	302
3.72.2.1	Foot_Water_Per_PSI	302
3.72.3	Function Documentation	303
3.72.3.1	MPX5010GP_Calculate_PSI	303
3.72.3.2	MPX5010GP_Depth	304
3.73	MPX5010GP.h	305
3.74	Testing/Test_Battery.c File Reference	305
3.74.1	Detailed Description	306
3.74.2	Macro Definition Documentation	306

3.74.2.1	F_CPU	306
3.74.3	Function Documentation	306
3.74.3.1	Test_Battery	306
3.75	Test_Battery.c	307
3.76	Testing/Test_Battery.h File Reference	308
3.76.1	Function Documentation	308
3.76.1.1	Test_Battery	308
3.77	Test_Battery.h	309
3.78	Testing/Test_H_Bridge.c File Reference	309
3.78.1	Detailed Description	310
3.78.2	Macro Definition Documentation	311
3.78.2.1	F_CPU	311
3.78.3	Function Documentation	311
3.78.3.1	Test_H_Bridge	311
3.78.3.2	Test_H_Bridge_Motor_ALL_Decrease	313
3.78.3.3	Test_H_Bridge_Motor_ALL_Increase	313
3.78.3.4	Test_H_Bridge_Motor_X_Decrease	314
3.78.3.5	Test_H_Bridge_Motor_X_Increase	315
3.78.3.6	Test_H_Bridge_Motor_Y_Decrease	316
3.78.3.7	Test_H_Bridge_Motor_Y_Increase	317
3.78.3.8	Test_H_Bridge_Motor_Z_Decrease	318
3.78.3.9	Test_H_Bridge_Motor_Z_Increase	319
3.79	Test_H_Bridge.c	320
3.80	Testing/Test_H_Bridge.h File Reference	323
3.80.1	Detailed Description	323
3.80.2	Function Documentation	324
3.80.2.1	Test_H_Bridge	324
3.81	Test_H_Bridge.h	326
3.82	Testing/Test_I2C_Compass.c File Reference	326
3.82.1	Detailed Description	326
3.82.2	Macro Definition Documentation	327
3.82.2.1	F_CPU	327
3.82.3	Function Documentation	327
3.82.3.1	Test_I2C_Compass	327
3.82.3.2	Test_I2C_Compass_Heading	328
3.82.3.3	Test_I2C_Compass_Mag_Acc	328
3.83	Test_I2C_Compass.c	329
3.84	Testing/Test_I2C_Compass.h File Reference	330
3.84.1	Detailed Description	331
3.84.2	Function Documentation	331

3.84.2.1	Test_I2C_Compass	331
3.85	Test_I2C_Compass.h	332
3.86	Testing/Test_LED.c File Reference	332
3.86.1	Detailed Description	333
3.86.2	Macro Definition Documentation	333
3.86.2.1	F_CPU	333
3.86.3	Function Documentation	333
3.86.3.1	Test_LED	333
3.86.3.2	Test_LED_Change_LED	334
3.86.3.3	Test_LED_Front_Toggle	334
3.87	Test_LED.c	335
3.88	Testing/Test_LED.h File Reference	336
3.88.1	Detailed Description	336
3.88.2	Function Documentation	336
3.88.2.1	Test_LED	336
3.89	Test_LED.h	337
3.90	Testing/Test_Motor_Control.c File Reference	337
3.90.1	Detailed Description	338
3.90.2	Macro Definition Documentation	339
3.90.2.1	F_CPU	339
3.90.2.2	Motor_Control_Test_Speed	339
3.90.3	Function Documentation	339
3.90.3.1	Test_Motor_Control	339
3.90.3.2	Test_Motor_Control_X_Constant	340
3.90.3.3	Test_Motor_Control_X_Varying	341
3.90.3.4	Test_Motor_Control_Y_Constant	342
3.90.3.5	Test_Motor_Control_Y_Varying	343
3.90.3.6	Test_Motor_Control_Z_Constant	344
3.90.3.7	Test_Motor_Control_Z_Varying	345
3.91	Test_Motor_Control.c	346
3.92	Testing/Test_Motor_Control.h File Reference	348
3.92.1	Detailed Description	349
3.92.2	Function Documentation	349
3.92.2.1	Test_Motor_Control	349
3.93	Test_Motor_Control.h	350
3.94	Testing/Test_Multiplexer.c File Reference	351
3.94.1	Detailed Description	351
3.94.2	Macro Definition Documentation	352
3.94.2.1	F_CPU	352
3.94.3	Function Documentation	352

3.94.3.1	Test_Multiplexer	352
3.94.3.2	Test_Multiplexer_Multiple_Pins	353
3.94.3.3	Test_Multiplexer_Single_Pin	353
3.94.3.4	Test_Multiplexer_Voltage_Conversion	354
3.94.3.5	Test_Multiplexer_Voltage_Multiple_Conversions	355
3.95	Test_Multiplexer.c	355
3.96	Testing/Test_Multiplexer.h File Reference	357
3.96.1	Detailed Description	358
3.96.2	Function Documentation	358
3.96.2.1	Test_Multiplexer	358
3.97	Test_Multiplexer.h	359
3.98	Testing/Test_Photodiodes.c File Reference	359
3.98.1	Detailed Description	360
3.98.2	Macro Definition Documentation	361
3.98.2.1	F_CPU	361
3.98.3	Function Documentation	361
3.98.3.1	Test_Photodiodes	361
3.98.3.2	Test_Photodiodes_All	362
3.98.3.3	Test_Photodiodes_All_With_LED	363
3.98.3.4	Test_Photodiodes_Front	364
3.98.3.5	Test_Photodiodes_Left	365
3.98.3.6	Test_Photodiodes_Print_Front	366
3.98.3.7	Test_Photodiodes_Print_Left	367
3.98.3.8	Test_Photodiodes_Print_Rear	368
3.98.3.9	Test_Photodiodes_Print_Right	368
3.98.3.10	Test_Photodiodes_Rear	369
3.98.3.11	Test_Photodiodes_Right	370
3.99	Test_Photodiodes.c	371
3.100	Testing/Test_Photodiodes.h File Reference	374
3.100.1	Detailed Description	374
3.100.2	Function Documentation	375
3.100.2.1	Test_Photodiodes	375
3.101	Test_Photodiodes.h	376
3.102	Testing/Test_Pressure_Sensor.c File Reference	376
3.102.1	Detailed Description	377
3.102.2	Macro Definition Documentation	377
3.102.2.1	F_CPU	377
3.102.3	Function Documentation	377
3.102.3.1	Test_Pressure_Sensor	377
3.103	Test_Pressure_Sensor.c	378

3.104 Testing/Test_Pressure_Sensor.h File Reference	379
3.104.1 Detailed Description	379
3.104.2 Function Documentation	380
3.104.2.1 Test_Pressure_Sensor	380
3.105 Test_Pressure_Sensor.h	381
3.106 Testing/Test_Swarming.c File Reference	381
3.106.1 Detailed Description	381
3.106.2 Macro Definition Documentation	382
3.106.2.1 F_CPU	382
3.106.3 Function Documentation	382
3.106.3.1 Test_Swarming	382
3.107 Test_Swarming.c	383
3.108 Testing/Test_Swarming.h File Reference	384
3.108.1 Detailed Description	385
3.108.2 Function Documentation	385
3.108.2.1 Test_Swarming	385
3.109 Test_Swarming.h	386

Chapter 1

Brief Description

The goal of this project was to build a swarm of autonomous robots to map underwater terrain. Research was conducted by the Autonomous Underwater Robots team to determine the best means in which to approach this problem. Specialized detection methods were generated by using blue LEDs and blue filtered photodiodes. The physical design of the robots involved using RC submarine platforms that were modified to include additional subsystems. The additional subsystems used for each submarine are the detection array, the power system, the motor control system, and the camera system. Individual swarm members were designed to swarm using minimalistic swarming techniques. They follow cohesion, alignment, and separation criteria in order to complete the task successfully. Image stitching software was used to compile an image from the smaller images taken by each of the swarm members. The swarm members will also be designed to meet specific cost criterion. Our proposed project can be accomplished with less than \$1000. The team investigated various cost-effective methods to waterproof the submarines. The autonomous submarines were designed with societal and environmental impacts taken into consideration.

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Main.c	This file is code main for testing and the main program	69
Main_Program.c	This file is code for the main program	72
Main_Program.h	This is the header file for the main program	75
Master_Build_Control.h	This is the header file to control which program builds	77
Algorithms/Motor_Control/Motor_Control.c	This file is code for the control loops for the three motors	7
Algorithms/Motor_Control/Motor_Control.h	This file is code for the control loops for the three motors	22
Algorithms/Navigation/Swarming.c	This file is code for the swarming algorithm	31
Algorithms/Navigation/Swarming.h	This file is the header file for the swarming algorithm	43
Algorithms/State_Machine/State_Machine.c	This file is code for the state machine code	48
Algorithms/State_Machine/State_Machine.h	This file is the header file for the state machine code	66
Microcontroller/ADC/ADC.c	This file is code to control the ADC on the ATmega328P	82
Microcontroller/ADC/ADC.h	This is the header file for the code that controls the ADC on the ATmega328P	88
Microcontroller/ADC/CD4051.c	This file is code to control the CD4051 multiplexer connected to the ADC	95
Microcontroller/ADC/CD4051.h	This file is code to control the CD4051 multiplexer connected to the ADC	101
Microcontroller/Communication/I2C_Interface.c	This file is code to control the TWI_Master	108
Microcontroller/Communication/I2C_Interface.h	This file is code to control the TWI_Master	116
Microcontroller/Communication/TWI_Master.c	This is a sample driver for the TWI hardware modules. It is interrupt driven. All functionality is controlled through passing information to and from functions	123
Microcontroller/Communication/TWI_Master.h	Header file for TWI_master.c . Include this file in the application	132

Microcontroller/Communication/ USART.c	146
This file is code to control the USART on the ATmega328P	
Microcontroller/Communication/ USART.h	159
This is the header file to control the USART on the ATmega328P	
Microcontroller/Interrupt/ Interrupt.c	169
This file is code for the interrupts on the ATmega328P	
Microcontroller/Interrupt/ Interrupt.h	172
Microcontroller/Interrupt/ Timer.c	172
This file is code to control the Timers on the ATmega328P	
Microcontroller/Interrupt/ Timer.h	179
This file is code to control the Timers on the ATmega328P	
Subsystem/H_Bridge/ DRV8830.c	183
This file is code to read control the h-bridges	
Subsystem/H_Bridge/ DRV8830.h	223
This file is code to read control the h-bridges	
Subsystem/IMU/ LSM303.c	237
This file is code to control the LSM303	
Subsystem/IMU/ LSM303.h	253
This file is code to control the LSM303	
Subsystem/Light/ CLS15.c	259
This file is code for the photodiode readings	
Subsystem/Light/ CLS15.h	274
This is the header file for the photodiode reading code	
Subsystem/Light/ XPEBBL.c	281
This file is code to control the 3 Watt LEDs	
Subsystem/Light/ XPEBBL.h	287
This file is code to control the 3 Watt LEDs	
Subsystem/Power/ Battery.c	290
This file is code to read the battery voltage	
Subsystem/Power/ Battery.h	294
This is the header file for the battery voltage reading code	
Subsystem/Pressure/ MPX5010GP.c	296
This file is code to read the depth from the MPX5010GP pressure sensor	
Subsystem/Pressure/ MPX5010GP.h	301
This file is code to read the depth from the MPX5010GP pressure sensor	
Testing/ Test_Battery.c	305
This file is for the testing of the battery code	
Testing/ Test_Battery.h	308
Testing/ Test_H_Bridge.c	309
This file is for the testing of the I2C H-Bridge code	
Testing/ Test_H_Bridge.h	323
This file is the header file for the testing of the H-Bridge code	
Testing/ Test_I2C_Compass.c	326
This file is for the testing of the I2C Compass code	
Testing/ Test_I2C_Compass.h	330
This is the header file for the testing of the I2C Compass code	
Testing/ Test_LED.c	332
This file is for the testing of the LED code	
Testing/ Test_LED.h	336
This is the header file for the testing of the LED code	
Testing/ Test_Motor_Control.c	337
This file is for the testing of the motor control code	
Testing/ Test_Motor_Control.h	348
This file is the header file for the testing of the motor control code	
Testing/ Test_Multiplexer.c	351
This file is for the testing of the multiplexer code	
Testing/ Test_Multiplexer.h	357
This file is the header file for the testing of the battery code	

Testing/ Test_Photodiodes.c	
This file is for the testing of the photodiode code	359
Testing/ Test_Photodiodes.h	
This is the header file for the testing of the photodiode code	374
Testing/ Test_Pressure_Sensor.c	
This file is for the testing of the pressure sensor code	376
Testing/ Test_Pressure_Sensor.h	
This is the header file for the testing of the pressure sensor code	379
Testing/ Test_Swarming.c	
This file is for the testing of the swarming code	381
Testing/ Test_Swarming.h	
This is the header file for the testing of the swarming code	384

Chapter 3

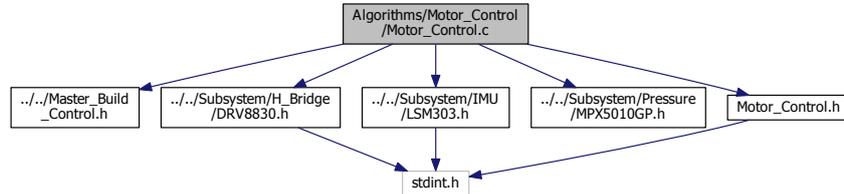
File Documentation

3.1 Algorithms/Motor_Control/Motor_Control.c File Reference

This file is code for the control loops for the three motors.

```
#include "../..../Master_Build_Control.h"  
#include "../..../Subsystem/H_Bridge/DRV8830.h"  
#include "../..../Subsystem/IMU/LSM303.h"  
#include "../..../Subsystem/Pressure/MPX5010GP.h"  
#include "Motor_Control.h"
```

Include dependency graph for Motor_Control.c:



Macros

- #define `motorZ_const` 1
The Z axis motor constant.
- #define `kpZ` 2.0
The proportional gain Z.
- #define `kiZ` 2.0
The integral gain Z.
- #define `kdZ` 1.0
The derivative gain Z.
- #define `Velocity_Time_Difference` 0.05
Time difference for the velocity calculation.
- #define `Motor_Z_Control_Type` 1
Motor control type for the Z motor.

Functions

- void [Motor_Control_X_Open](#) ()
- void [Motor_Control_Y_Open](#) ()
- void [Motor_Control_Z](#) (float depth)
- void [Motor_Control](#) (void)
This function is the control loops for the three motors.
- void [Motor_Control_Update_Velocity_X](#) (float Updated_Value)
This function is the desired velocity for the X-axis motor.
- void [Motor_Control_Update_Velocity_Y](#) (float Updated_Value)
This function is the desired velocity for the Y-axis motor.
- void [Motor_Control_Update_Depth_Z](#) (float Updated_Value)
This function is the desired depth of the submarine.
- void [Motor_Control_Motor_X_Flag_Update](#) (uint8_t Input)
This function controls the value for the update control motor x flag.
- void [Motor_Control_Motor_Y_Flag_Update](#) (uint8_t Input)
This function controls the value for the update control motor y flag.
- void [Motor_Control_Motor_Z_Flag_Update](#) (uint8_t Input)
This function controls the value for the update control motor z flag.
- void [Motor_Control_Update](#) (void)
This function is used to update the motor control if the update flag is enabled.
- void [Motor_Control_Enable_Flag](#) (void)
This function is used to enable the control flag.

3.1.1 Detailed Description

This file is code for the control loops for the three motors.

Author

Ryan Lipski, Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 1/28/2015
Created: 1/27/2015 4:45:33 PM

Definition in file [Motor_Control.c](#).

3.1.2 Macro Definition Documentation

3.1.2.1 #define kdZ 1.0

The derivative gain Z.

Definition at line 16 of file [Motor_Control.c](#).

Referenced by [Motor_Control_Z\(\)](#).

3.1.2.2 #define kiZ 2.0

The integral gain Z.

Definition at line 14 of file [Motor_Control.c](#).

Referenced by [Motor_Control_Z\(\)](#).

3.1.2.3 #define kpZ 2.0

The proportional gain Z.

Definition at line 12 of file [Motor_Control.c](#).

Referenced by [Motor_Control_Z\(\)](#).

3.1.2.4 #define Motor_Z_Control_Type 1

Motor control type for the Z motor.

Definition at line 20 of file [Motor_Control.c](#).

3.1.2.5 #define motorZ_const 1

The Z axis motor constant.

Definition at line 10 of file [Motor_Control.c](#).

Referenced by [Motor_Control_Z\(\)](#).

3.1.2.6 #define Velocity_Time_Difference 0.05

Time difference for the velocity calculation.

Definition at line 18 of file [Motor_Control.c](#).

3.1.3 Function Documentation

3.1.3.1 void Motor_Control (void)

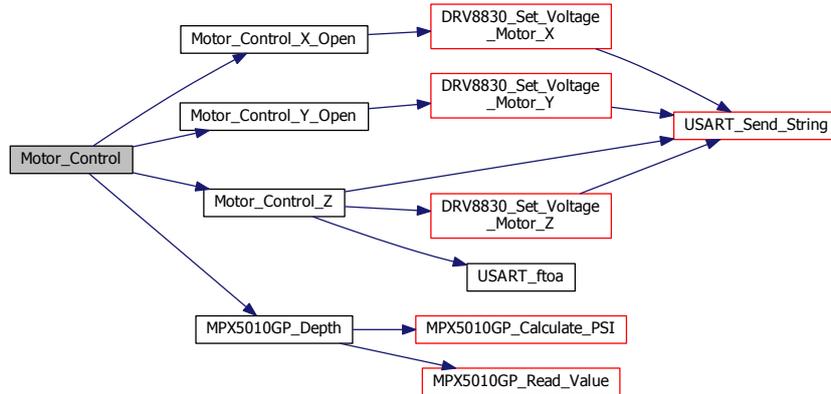
This function is the control loops for the three motors.

Definition at line 95 of file [Motor_Control.c](#).

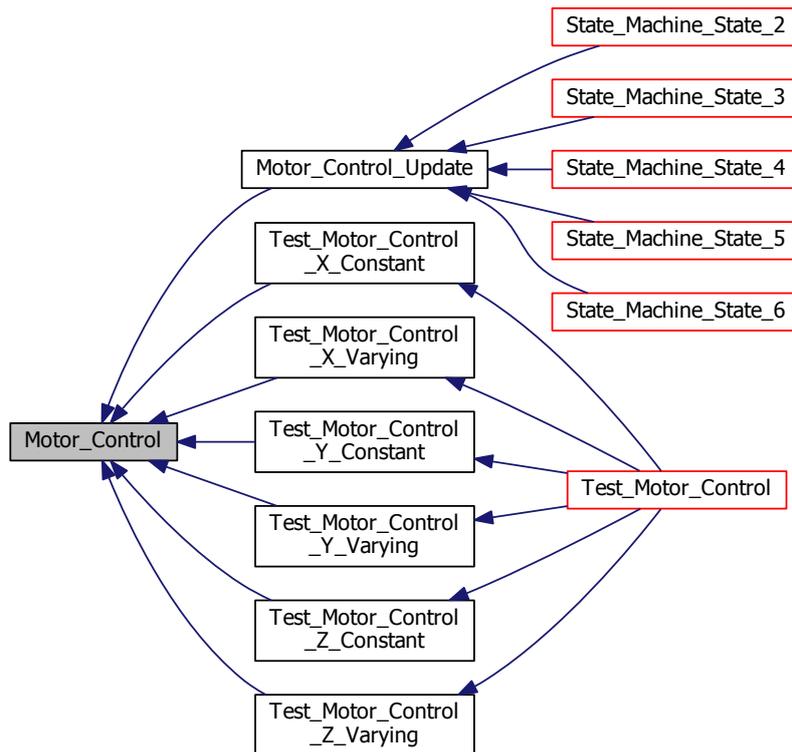
References [Motor_Control_X_Open\(\)](#), [Motor_Control_Y_Open\(\)](#), [Motor_Control_Z\(\)](#), and [MPX5010GP_Depth\(\)](#).

Referenced by [Motor_Control_Update\(\)](#), [Test_Motor_Control_X_Constant\(\)](#), [Test_Motor_Control_X_Varying\(\)](#), [Test_Motor_Control_Y_Constant\(\)](#), [Test_Motor_Control_Y_Varying\(\)](#), [Test_Motor_Control_Z_Constant\(\)](#), and [Test_Motor_Control_Z_Varying\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



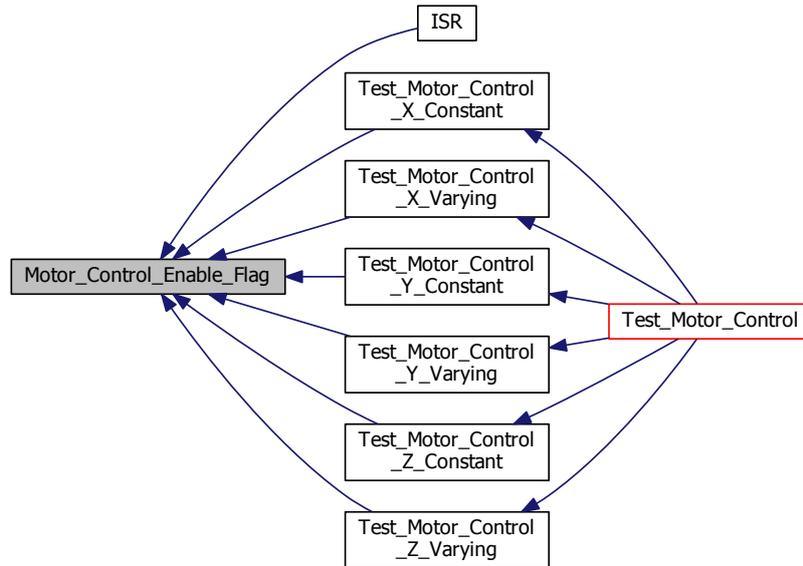
3.1.3.2 void Motor_Control_Enable_Flag (void)

This function is used to enable the control flag.

Definition at line 180 of file [Motor_Control.c](#).

Referenced by [ISR\(\)](#), [Test_Motor_Control_X_Constant\(\)](#), [Test_Motor_Control_X_Varying\(\)](#), [Test_Motor_Control_Y_Constant\(\)](#), [Test_Motor_Control_Y_Varying\(\)](#), [Test_Motor_Control_Z_Constant\(\)](#), and [Test_Motor_Control_Z_Varying\(\)](#).

Here is the caller graph for this function:



3.1.3.3 void Motor_Control_Motor_X_Flag_Update (uint8_t Input)

This function controls the value for the update control motor x flag.

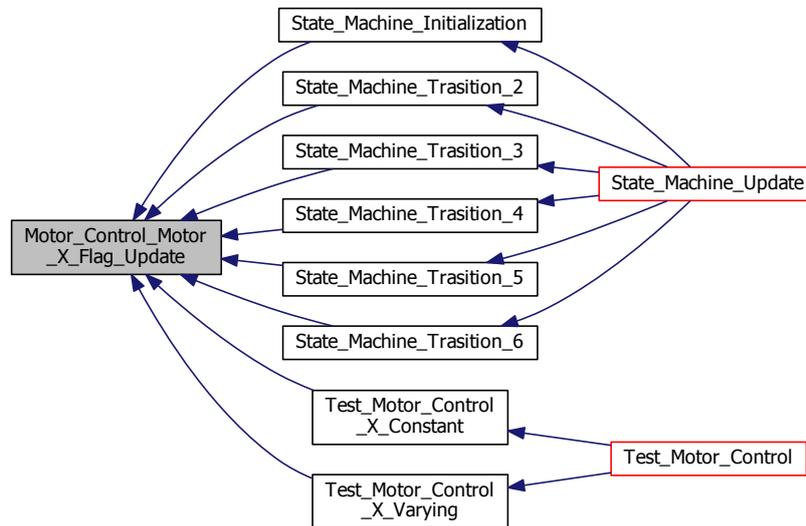
Parameters

<i>Input</i>	Value to set the update control motor x flag.
--------------	---

Definition at line 142 of file [Motor_Control.c](#).

Referenced by [State_Machine_Initialization\(\)](#), [State_Machine_Trasition_2\(\)](#), [State_Machine_Trasition_3\(\)](#), [State_Machine_Trasition_4\(\)](#), [State_Machine_Trasition_5\(\)](#), [State_Machine_Trasition_6\(\)](#), [Test_Motor_Control_X_Constant\(\)](#), and [Test_Motor_Control_X_Varying\(\)](#).

Here is the caller graph for this function:



3.1.3.4 void Motor_Control_Motor_Y_Flag_Update (uint8_t Input)

This function controls the value for the update control motor y flag.

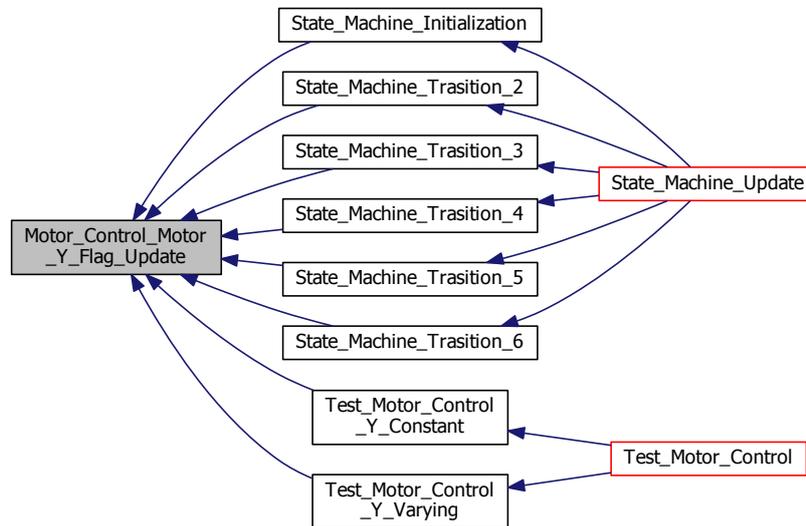
Parameters

<i>Input</i>	Value to set the update control motor y flag.
--------------	---

Definition at line 151 of file [Motor_Control.c](#).

Referenced by [State_Machine_Initialization\(\)](#), [State_Machine_Trasition_2\(\)](#), [State_Machine_Trasition_3\(\)](#), [State_Machine_Trasition_4\(\)](#), [State_Machine_Trasition_5\(\)](#), [State_Machine_Trasition_6\(\)](#), [Test_Motor_Control_Y_Constant\(\)](#), and [Test_Motor_Control_Y_Varying\(\)](#).

Here is the caller graph for this function:



3.1.3.5 void Motor_Control_Motor_Z_Flag_Update (uint8_t Input)

This function controls the value for the update control motor z flag.

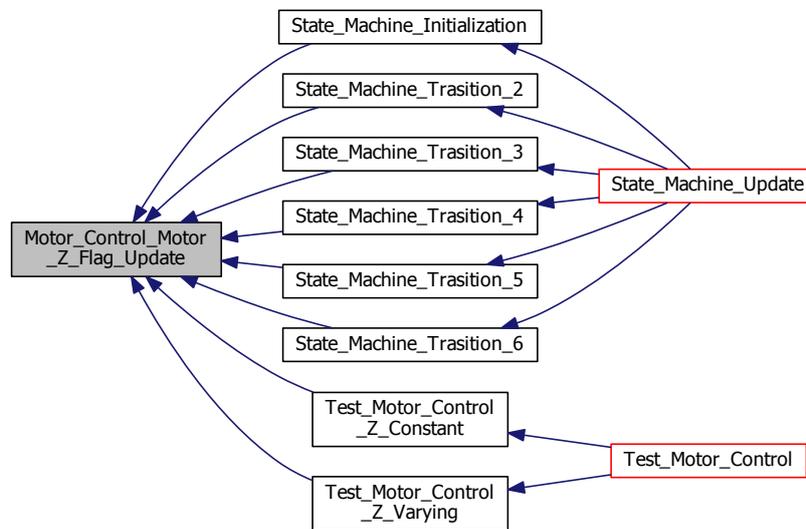
Parameters

<i>Input</i>	Value to set the update control motor z flag.
--------------	---

Definition at line 160 of file [Motor_Control.c](#).

Referenced by [State_Machine_Initialization\(\)](#), [State_Machine_Trasition_2\(\)](#), [State_Machine_Trasition_3\(\)](#), [State_Machine_Trasition_4\(\)](#), [State_Machine_Trasition_5\(\)](#), [State_Machine_Trasition_6\(\)](#), [Test_Motor_Control_Z_Constant\(\)](#), and [Test_Motor_Control_Z_Varying\(\)](#).

Here is the caller graph for this function:



3.1.3.6 void Motor_Control_Update (void)

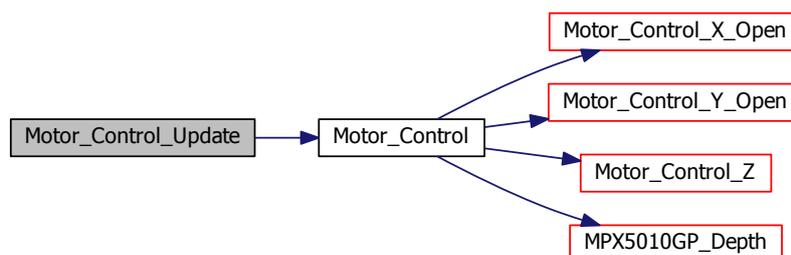
This function is used to update the motor control if the update flag is enabled.

Definition at line 168 of file [Motor_Control.c](#).

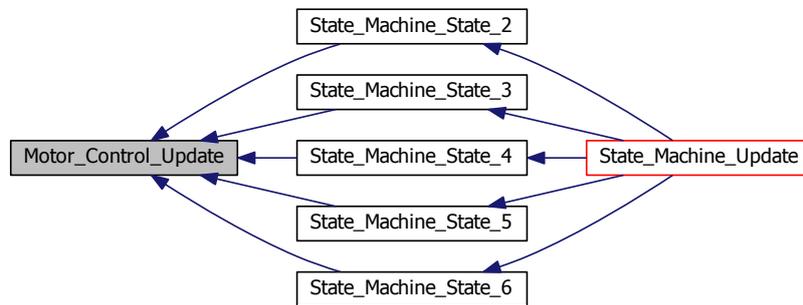
References [Motor_Control\(\)](#).

Referenced by [State_Machine_State_2\(\)](#), [State_Machine_State_3\(\)](#), [State_Machine_State_4\(\)](#), [State_Machine_State_5\(\)](#), and [State_Machine_State_6\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.3.7 void Motor_Control_Update_Depth_Z (float Updated_Value)

This function is the desired depth of the submarine.

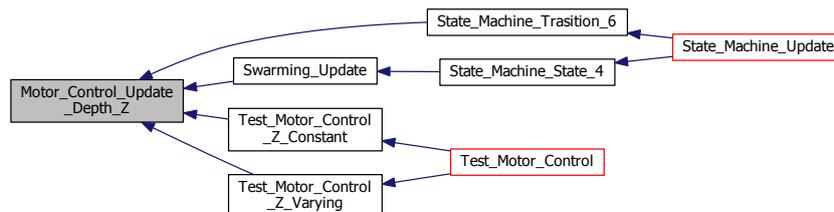
Parameters

Updated_Value	The new desired depth for the submarine.
---------------	--

Definition at line 133 of file Motor_Control.c.

Referenced by State_Machine_Trasition_6(), Swarming_Update(), Test_Motor_Control_Z_Constant(), and Test_Motor_Control_Z_Varying().

Here is the caller graph for this function:



3.1.3.8 void Motor_Control_Update_Velocity_X (float Updated_Value)

This function is the desired velocity for the X-axis motor.

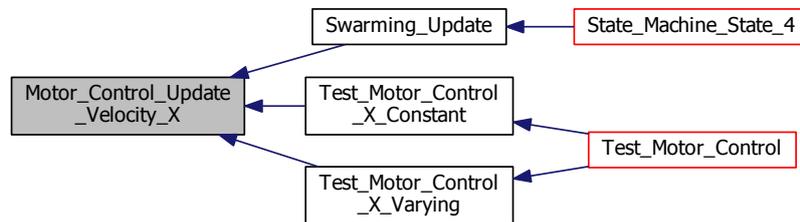
Parameters

Updated_Value	The new desired velocity for the X-axis motor.
---------------	--

Definition at line 115 of file Motor_Control.c.

Referenced by Swarming_Update(), Test_Motor_Control_X_Constant(), and Test_Motor_Control_X_Varying().

Here is the caller graph for this function:



3.1.3.9 void Motor_Control_Update_Velocity_Y (float Updated_Value)

This function is the desired velocity for the Y-axis motor.

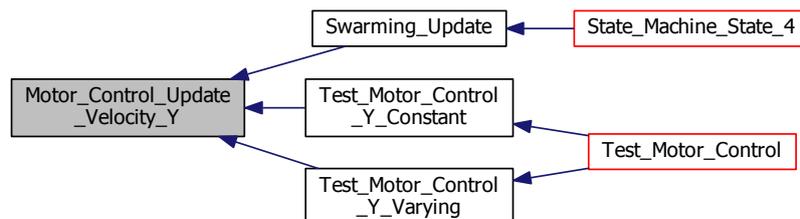
Parameters

<i>Updated_Value</i>	The new desired velocity for the Y-axis motor.
----------------------	--

Definition at line 124 of file [Motor_Control.c](#).

Referenced by [Swarming_Update\(\)](#), [Test_Motor_Control_Y_Constant\(\)](#), and [Test_Motor_Control_Y_Varying\(\)](#).

Here is the caller graph for this function:



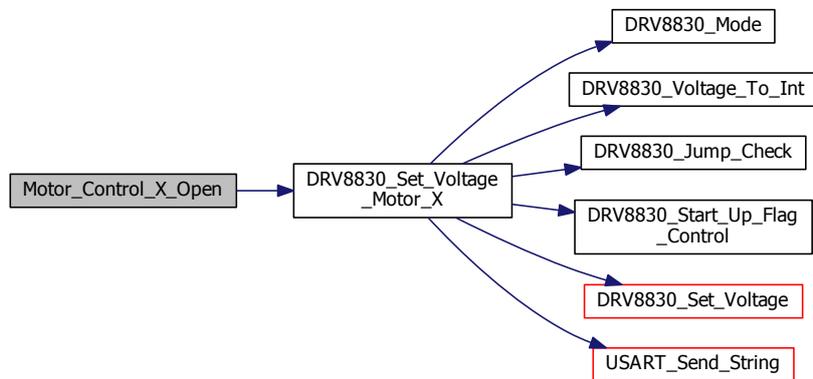
3.1.3.10 void Motor_Control_X_Open ()

Definition at line 44 of file [Motor_Control.c](#).

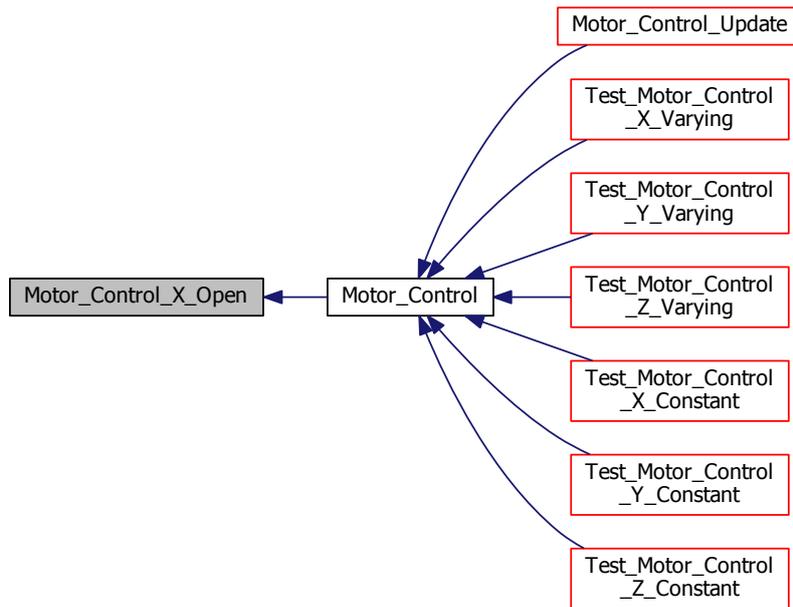
References [DRV8830_Set_Voltage_Motor_X\(\)](#), and [Motor_Max_Voltage](#).

Referenced by [Motor_Control\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



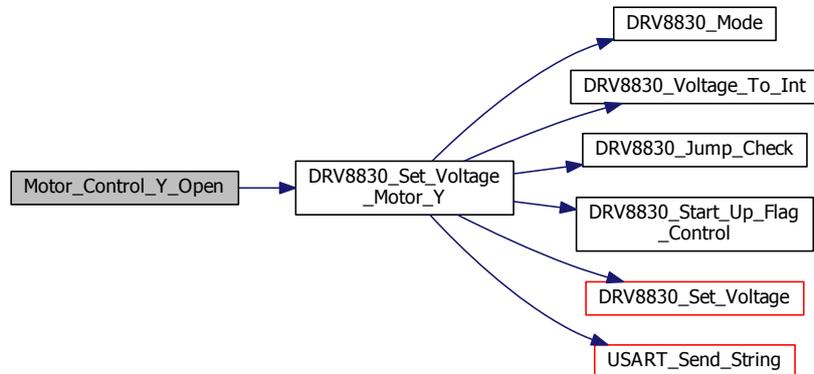
3.1.3.11 void Motor_Control_Y_Open ()

Definition at line 49 of file [Motor_Control.c](#).

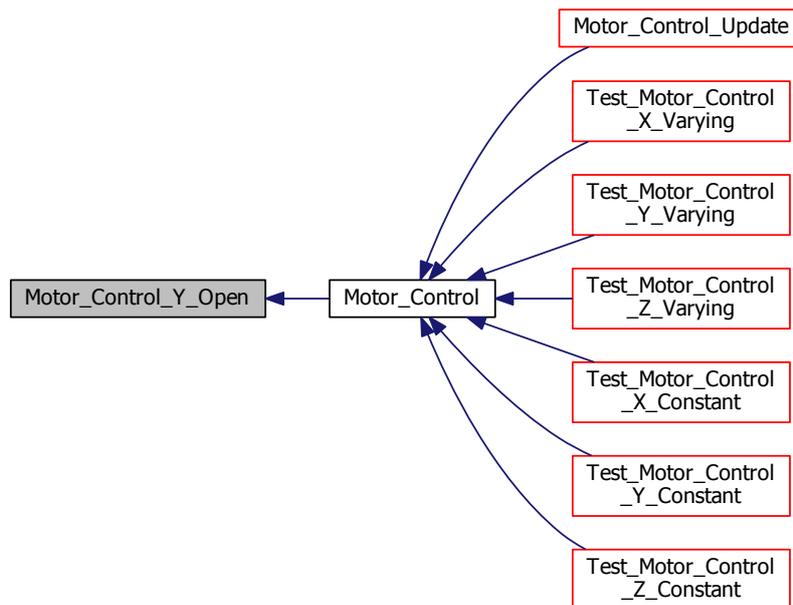
References [DRV8830_Set_Voltage_Motor_Y\(\)](#), and [Motor_Max_Voltage](#).

Referenced by [Motor_Control\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



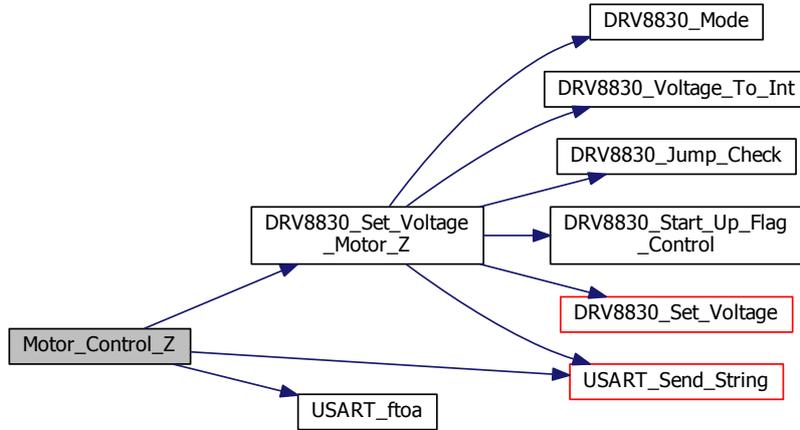
3.1.3.12 void Motor_Control_Z (float *depth*)

Definition at line 54 of file [Motor_Control.c](#).

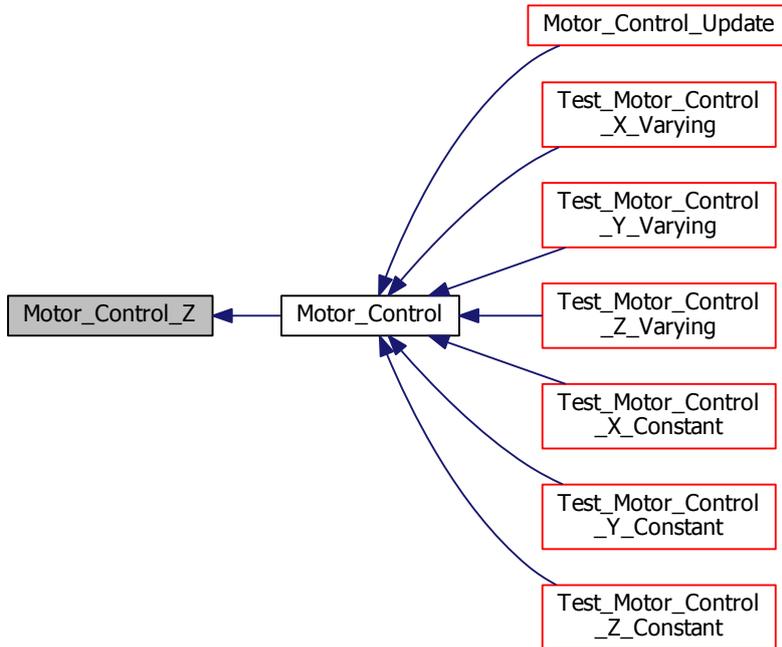
References [DRV8830_Set_Voltage_Motor_Z\(\)](#), [FLOATING_POINT_BUFFER_SIZE](#), [kdZ](#), [kiZ](#), [kpZ](#), [Motor_Max_Voltage](#), [motorZ_const](#), [USART_ftoa\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Motor_Control\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.2 Motor_Control.c

```

00001 /**
00002  * @file Motor_Control.c
00003  * @author Ryan Lipski, Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 1/28/2015\n Created: 1/27/2015 4:45:33 PM
    
```

```

00006  * @brief This file is code for the control loops for the three motors.
00007  */
00008
00009  /** @brief The Z axis motor constant. */
00010  #define motorZ_const 1
00011  /** @brief The proportional gain Z. */
00012  #define kpZ 2.0
00013  /** @brief The integral gain Z. */
00014  #define kiZ 2.0
00015  /** @brief The derivative gain Z. */
00016  #define kdZ 1.0
00017  /** @brief Time difference for the velocity calculation. */
00018  #define Velocity_Time_Difference 0.05
00019  /** @brief Motor control type for the Z motor. */
00020  #define Motor_Z_Control_Type 1
00021
00022  #include "../Master_Build_Control.h"
00023  #include "../Subsystem/H_Bridge/DRV8830.h"
00024  #include "../Subsystem/IMU/LSM303.h"
00025  #include "../Subsystem/Pressure/MPX5010GP.h"
00026  #include "Motor_Control.h"
00027
00028  #if Print_Motor_Information == 1
00029      #include <stdio.h>
00030      #include "../Microcontroller/Communication/USART.h"
00031  #endif
00032
00033  /* @brief Desired velocity of the x-axis motor. */
00034  static volatile float v_desX = 0.0;
00035  /* @brief Desired velocity of the x-axis motor. */
00036  static volatile float v_desY = 0.0;
00037  /* @brief Desired depth of the submarine. */
00038  static volatile float p_desZ = 2.0;
00039  static uint8_t Update_Control_Flag = 1;
00040  static uint8_t Update_Motor_X_Flag = 0;
00041  static uint8_t Update_Motor_Y_Flag = 0;
00042  static uint8_t Update_Motor_Z_Flag = 0;
00043
00044  void Motor_Control_X_Open()
00045  {
00046      DRV8830_Set_Voltage_Motor_X(v_desX/5.0*
Motor_Max_Voltage);
00047  }
00048
00049  void Motor_Control_Y_Open()
00050  {
00051      DRV8830_Set_Voltage_Motor_Y(v_desY/5.0*
Motor_Max_Voltage);
00052  }
00053
00054  void Motor_Control_Z(float depth)
00055  {
00056      float errZ = p_desZ - depth; //calculate the error in position
00057      #if Motor_Z_Control_Type == 0
00058          if(errZ>0.1)
00059          {
00060              DRV8830_Set_Voltage_Motor_Z(
Motor_Max_Voltage/2);
00061          }
00062          else if(errZ<-0.1)
00063          {
00064              DRV8830_Set_Voltage_Motor_Z(-
Motor_Max_Voltage);
00065          }
00066          else
00067          {
00068              DRV8830_Set_Voltage_Motor_Z(-
Motor_Max_Voltage/2);
00069          }
00070      #elif Motor_Z_Control_Type == 1
00071          if(errZ<0.1 && errZ>-0.1)//dead band for depth
00072          {
00073              errZ=0;
00074          }
00075          static float errZ_old = 0;//old error signal for motor 1
00076          static float errZ_old_2 = 0;//old error signal for motor
00077          float rZ = kpZ*(errZ-errZ_old_2)+kiZ*(errZ+errZ_old_2)/2+kdZ*(errZ-2*errZ_old+errZ_old_2);
00078          errZ_old_2 = errZ_old;
00079          errZ_old = errZ;
00080          DRV8830_Set_Voltage_Motor_Z(rZ*motorZ_const);//set the motor
3's voltage using 5.14 * (VSET)/64 = Vout
00081          #if Print_Motor_Information == 1
00082              char Buffer[100] = {0x00};
00083              char Buffer_Depth[FLOATING_POINT_BUFFER_SIZE] = {0x00};
00084              char Buffer_Error[FLOATING_POINT_BUFFER_SIZE] = {0x00};
00085              char Buffer_Voltage[FLOATING_POINT_BUFFER_SIZE] = {0x00};
00086              sprintf(Buffer,"Depth: %s errZ %s zvoltage %s\r\n", USART_ftoa(depth, Buffer_Depth),

```

```

    USART_ftoa(errZ, Buffer_Error), USART_ftoa(rZ*motorZ_const, Buffer_Voltage));
00087     USART_Send_String(Buffer);
00088     #endif
00089     #endif
00090 }
00091
00092 /**
00093  * @brief This function is the control loops for the three motors.
00094  */
00095 void Motor_Control(void)
00096 {
00097     if(Update_Motor_X_Flag)
00098     {
00099         Motor_Control_X_Open();
00100     }
00101     if(Update_Motor_Y_Flag)
00102     {
00103         Motor_Control_Y_Open();
00104     }
00105     if(Update_Motor_Z_Flag)
00106     {
00107         Motor_Control_Z(MPX5010GP_Depth());
00108     }
00109 }
00110
00111 /**
00112  * @brief This function is the desired velocity for the X-axis motor.
00113  * @param Updated_Value The new desired velocity for the X-axis motor.
00114  */
00115 void Motor_Control_Update_Velocity_X(float Updated_Value)
00116 {
00117     v_desX = Updated_Value;
00118 }
00119
00120 /**
00121  * @brief This function is the desired velocity for the Y-axis motor.
00122  * @param Updated_Value The new desired velocity for the Y-axis motor.
00123  */
00124 void Motor_Control_Update_Velocity_Y(float Updated_Value)
00125 {
00126     v_desY = Updated_Value;
00127 }
00128
00129 /**
00130  * @brief @brief This function is the desired depth of the submarine.
00131  * @param Updated_Value The new desired depth for the submarine.
00132  */
00133 void Motor_Control_Update_Depth_Z(float Updated_Value)
00134 {
00135     p_desZ = Updated_Value;
00136 }
00137
00138 /**
00139  * @brief This function controls the value for the update control motor x flag.
00140  * @param Input Value to set the update control motor x flag.
00141  */
00142 void Motor_Control_Motor_X_Flag_Update(uint8_t Input)
00143 {
00144     Update_Motor_X_Flag = Input;
00145 }
00146
00147 /**
00148  * @brief This function controls the value for the update control motor y flag.
00149  * @param Input Value to set the update control motor y flag.
00150  */
00151 void Motor_Control_Motor_Y_Flag_Update(uint8_t Input)
00152 {
00153     Update_Motor_Y_Flag = Input;
00154 }
00155
00156 /**
00157  * @brief This function controls the value for the update control motor z flag.
00158  * @param Input Value to set the update control motor z flag.
00159  */
00160 void Motor_Control_Motor_Z_Flag_Update(uint8_t Input)
00161 {
00162     Update_Motor_Z_Flag = Input;
00163 }
00164
00165 /**
00166  * @brief This function is used to update the motor control if the update flag is enabled.
00167  */
00168 void Motor_Control_Update(void)
00169 {
00170     if(Update_Control_Flag)
00171     {
00172         Motor_Control();

```

```

00173     Update_Control_Flag = 0;
00174     }
00175 }
00176
00177 /**
00178  * @brief This function is used to enable the control flag.
00179  */
00180 void Motor_Control_Enable_Flag(void)
00181 {
00182     Update_Control_Flag = 1;
00183 }

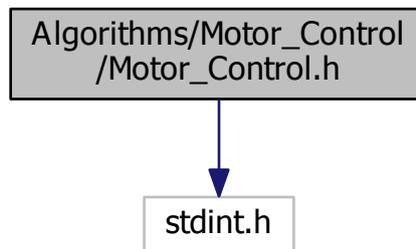
```

3.3 Algorithms/Motor_Control/Motor_Control.h File Reference

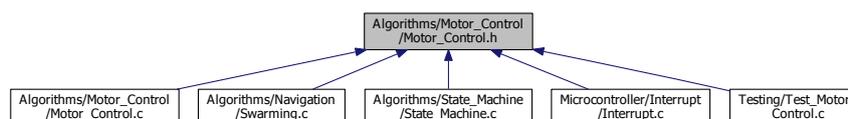
This file is code for the control loops for the three motors.

```
#include <stdint.h>
```

Include dependency graph for Motor_Control.h:



This graph shows which files directly or indirectly include this file:



Functions

- void [Motor_Control](#) (void)
This function is the control loops for the three motors.
- void [Motor_Control_Update_Velocity_X](#) (float UpdateValue)
This function is the desired velocity for the X-axis motor.
- void [Motor_Control_Update_Velocity_Y](#) (float Updated_Value)
This function is the desired velocity for the Y-axis motor.
- void [Motor_Control_Update_Depth_Z](#) (float UpdateValue)
This function is the desired depth of the submarine.
- void [Motor_Control_Motor_X_Flag_Update](#) (uint8_t Input)
This function controls the value for the update control motor x flag.

- void [Motor_Control_Motor_Y_Flag_Update](#) (uint8_t Input)

This function controls the value for the update control motor y flag.

- void [Motor_Control_Motor_Z_Flag_Update](#) (uint8_t Input)

This function controls the value for the update control motor z flag.

- void [Motor_Control_Update](#) (void)

This function is used to update the motor control if the update flag is enabled.

- void [Motor_Control_Enable_Flag](#) (void)

This function is used to enable the control flag.

3.3.1 Detailed Description

This file is code for the control loops for the three motors.

Author

Ryan Lipski, Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 1/28/2015

Created: 1/27/2015 4:45:50 PM

Definition in file [Motor_Control.h](#).

3.3.2 Function Documentation

3.3.2.1 void Motor_Control (void)

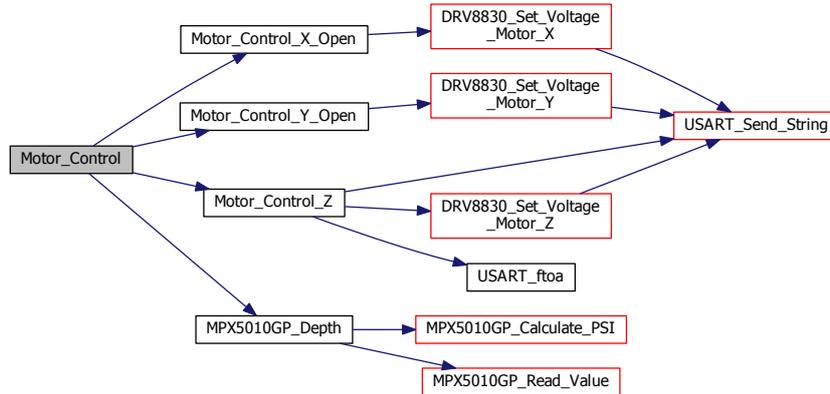
This function is the control loops for the three motors.

Definition at line 95 of file [Motor_Control.c](#).

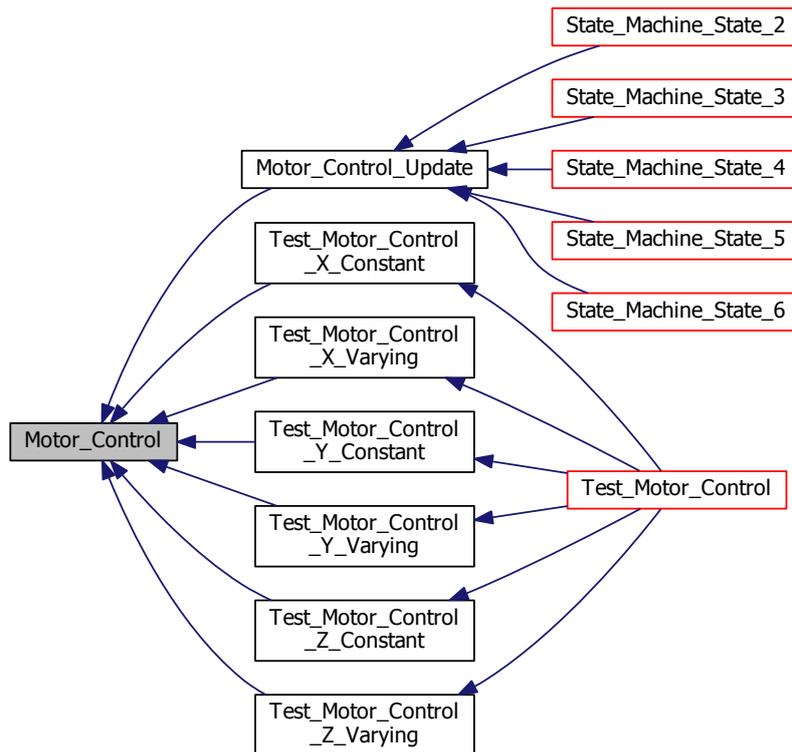
References [Motor_Control_X_Open\(\)](#), [Motor_Control_Y_Open\(\)](#), [Motor_Control_Z\(\)](#), and [MPX5010GP_Depth\(\)](#).

Referenced by [Motor_Control_Update\(\)](#), [Test_Motor_Control_X_Constant\(\)](#), [Test_Motor_Control_X_Varying\(\)](#), [Test_Motor_Control_Y_Constant\(\)](#), [Test_Motor_Control_Y_Varying\(\)](#), [Test_Motor_Control_Z_Constant\(\)](#), and [Test_Motor_Control_Z_Varying\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



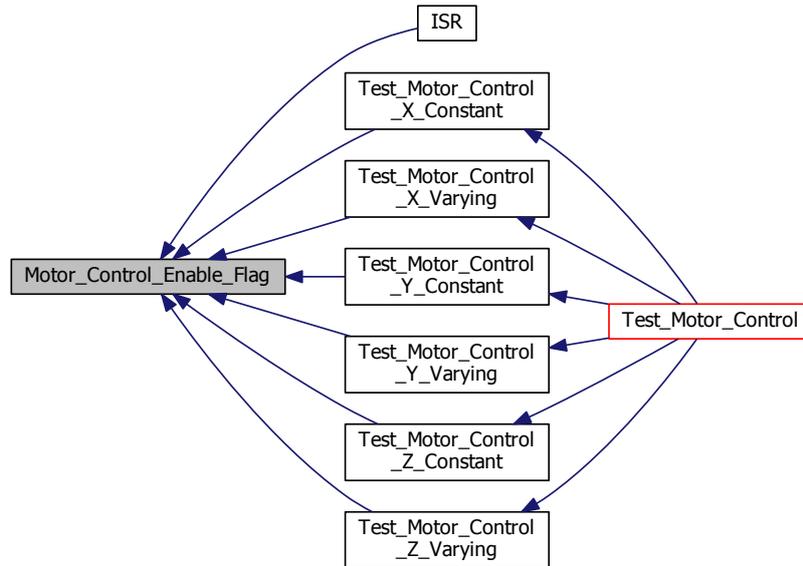
3.3.2.2 void Motor_Control_Enable_Flag (void)

This function is used to enable the control flag.

Definition at line 180 of file [Motor_Control.c](#).

Referenced by [ISR\(\)](#), [Test_Motor_Control_X_Constant\(\)](#), [Test_Motor_Control_X_Varying\(\)](#), [Test_Motor_Control_Y_Constant\(\)](#), [Test_Motor_Control_Y_Varying\(\)](#), [Test_Motor_Control_Z_Constant\(\)](#), and [Test_Motor_Control_Z_Varying\(\)](#).

Here is the caller graph for this function:



3.3.2.3 void Motor_Control_Motor_X_Flag_Update (uint8_t Input)

This function controls the value for the update control motor x flag.

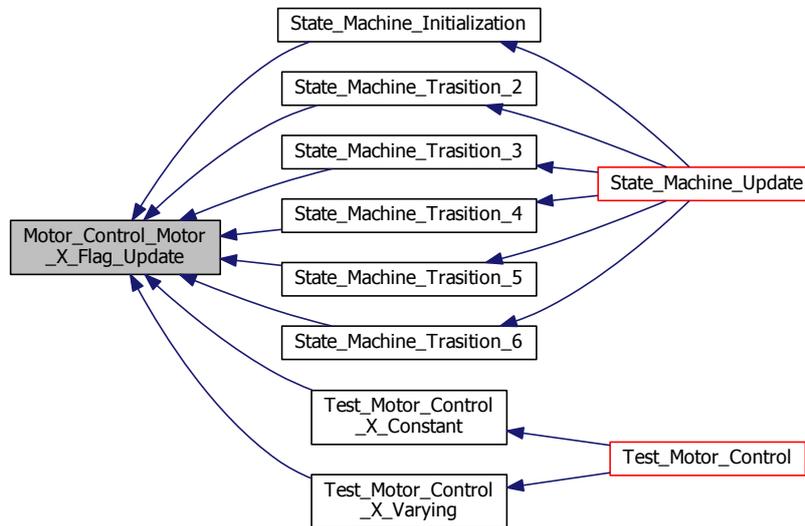
Parameters

<i>Input</i>	Value to set the update control motor x flag.
--------------	---

Definition at line 142 of file [Motor_Control.c](#).

Referenced by [State_Machine_Initialization\(\)](#), [State_Machine_Trasition_2\(\)](#), [State_Machine_Trasition_3\(\)](#), [State_Machine_Trasition_4\(\)](#), [State_Machine_Trasition_5\(\)](#), [State_Machine_Trasition_6\(\)](#), [Test_Motor_Control_X_Constant\(\)](#), and [Test_Motor_Control_X_Varying\(\)](#).

Here is the caller graph for this function:



3.3.2.4 void Motor_Control_Motor_Y_Flag_Update (uint8_t Input)

This function controls the value for the update control motor y flag.

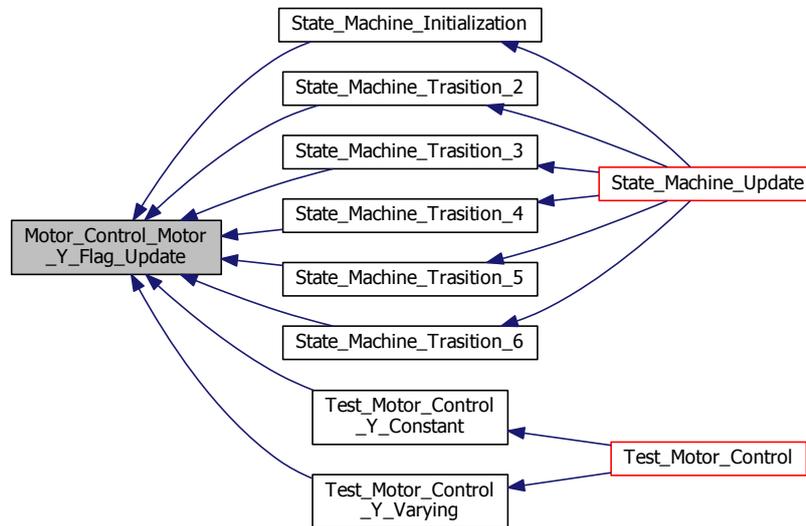
Parameters

<i>Input</i>	Value to set the update control motor y flag.
--------------	---

Definition at line 151 of file [Motor_Control.c](#).

Referenced by [State_Machine_Initialization\(\)](#), [State_Machine_Trasition_2\(\)](#), [State_Machine_Trasition_3\(\)](#), [State_Machine_Trasition_4\(\)](#), [State_Machine_Trasition_5\(\)](#), [State_Machine_Trasition_6\(\)](#), [Test_Motor_Control_Y_Constant\(\)](#), and [Test_Motor_Control_Y_Varying\(\)](#).

Here is the caller graph for this function:



3.3.2.5 void Motor_Control_Motor_Z_Flag_Update (uint8_t Input)

This function controls the value for the update control motor z flag.

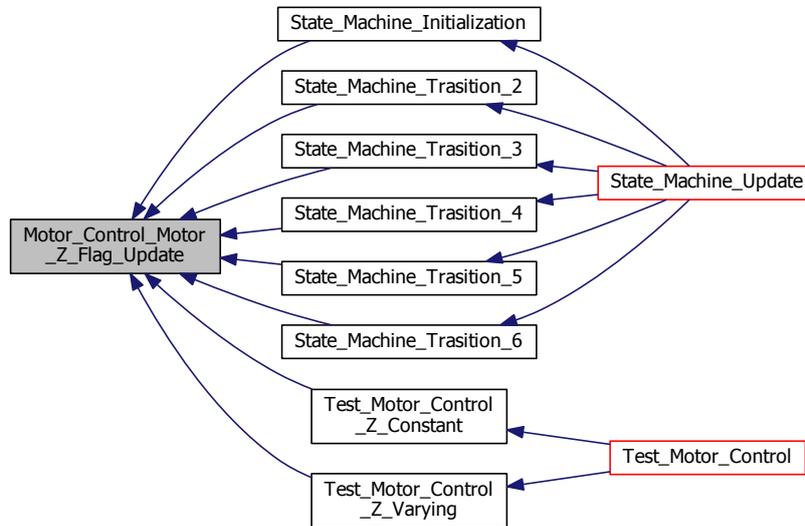
Parameters

<i>Input</i>	Value to set the update control motor z flag.
--------------	---

Definition at line 160 of file [Motor_Control.c](#).

Referenced by [State_Machine_Initialization\(\)](#), [State_Machine_Trasition_2\(\)](#), [State_Machine_Trasition_3\(\)](#), [State_Machine_Trasition_4\(\)](#), [State_Machine_Trasition_5\(\)](#), [State_Machine_Trasition_6\(\)](#), [Test_Motor_Control_Z_Constant\(\)](#), and [Test_Motor_Control_Z_Varying\(\)](#).

Here is the caller graph for this function:



3.3.2.6 void Motor_Control_Update (void)

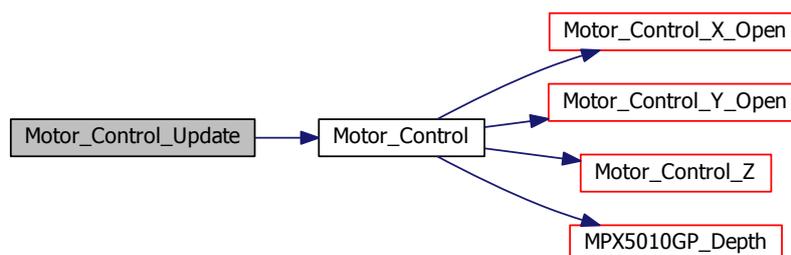
This function is used to update the motor control if the update flag is enabled.

Definition at line 168 of file [Motor_Control.c](#).

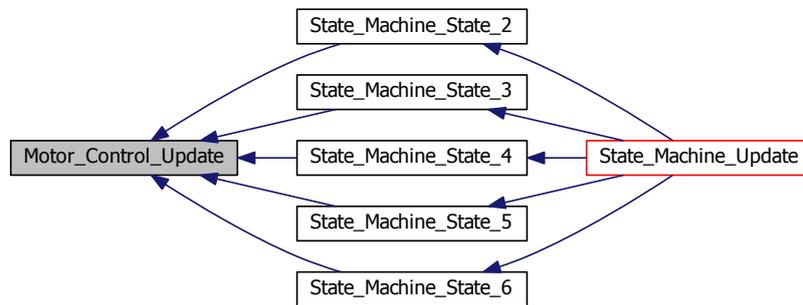
References [Motor_Control\(\)](#).

Referenced by [State_Machine_State_2\(\)](#), [State_Machine_State_3\(\)](#), [State_Machine_State_4\(\)](#), [State_Machine_State_5\(\)](#), and [State_Machine_State_6\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.3.2.7 void Motor_Control_Update_Depth_Z (float Updated_Value)

This function is the desired depth of the submarine.

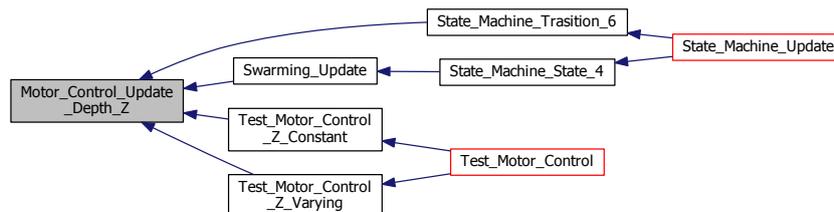
Parameters

Updated_Value	The new desired depth for the submarine.
---------------	--

Definition at line 133 of file Motor_Control.c.

Referenced by State_Machine_Trasition_6(), Swarming_Update(), Test_Motor_Control_Z_Constant(), and Test_Motor_Control_Z_Varying().

Here is the caller graph for this function:



3.3.2.8 void Motor_Control_Update_Velocity_X (float Updated_Value)

This function is the desired velocity for the X-axis motor.

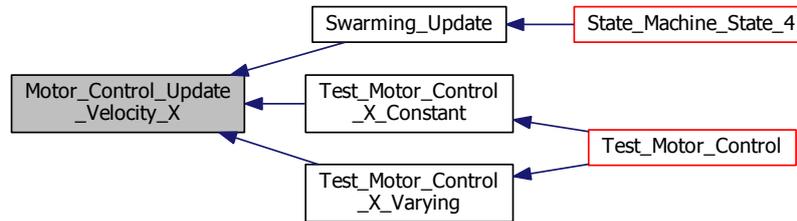
Parameters

Updated_Value	The new desired velocity for the X-axis motor.
---------------	--

Definition at line 115 of file Motor_Control.c.

Referenced by Swarming_Update(), Test_Motor_Control_X_Constant(), and Test_Motor_Control_X_Varying().

Here is the caller graph for this function:



3.3.2.9 void Motor_Control_Update_Velocity_Y (float Updated_Value)

This function is the desired velocity for the Y-axis motor.

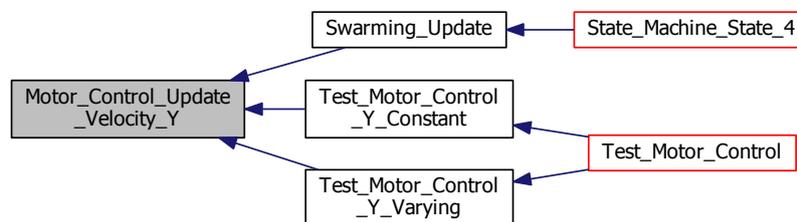
Parameters

<i>Updated_Value</i>	The new desired velocity for the Y-axis motor.
----------------------	--

Definition at line 124 of file [Motor_Control.c](#).

Referenced by [Swarming_Update\(\)](#), [Test_Motor_Control_Y_Constant\(\)](#), and [Test_Motor_Control_Y_Varying\(\)](#).

Here is the caller graph for this function:



3.4 Motor_Control.h

```

00001 /**
00002  * @file Motor_Control.h
00003  * @author Ryan Lipski, Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 1/28/2015\n Created: 1/27/2015 4:45:50 PM
00006  * @brief This file is code for the control loops for the three motors.
00007  */
00008
00009 #ifndef MOTOR_CONTROL_H_
00010 #define MOTOR_CONTROL_H_
00011
00012 #include <stdint.h>
00013
00014 void Motor_Control(void);
00015 void Motor_Control_Update_Velocity_X(float UpdateValue);
00016 void Motor_Control_Update_Velocity_Y(float Updated_Value);
00017 void Motor_Control_Update_Depth_Z(float UpdateValue);
00018 void Motor_Control_Motor_X_Flag_Update(uint8_t Input);
00019 void Motor_Control_Motor_Y_Flag_Update(uint8_t Input);
  
```

```

00020 void Motor_Control_Motor_Z_Flag_Update(uint8_t Input);
00021 void Motor_Control_Update(void);
00022 void Motor_Control_Enable_Flag(void);
00023
00024 #endif /* MOTOR_CONTROL_H_ */

```

3.5 Algorithms/Navigation/Swarming.c File Reference

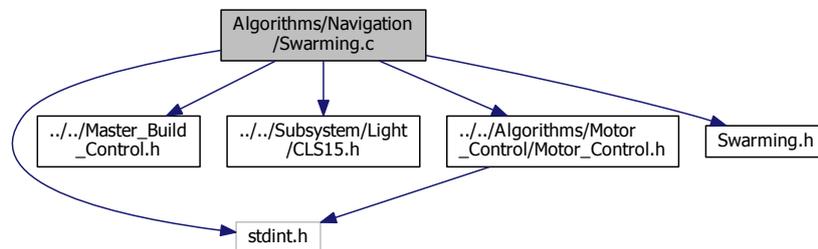
This file is code for the swarming algorithm.

```

#include <stdint.h>
#include "../..//Master_Build_Control.h"
#include "../..//Subsystem/Light/CLS15.h"
#include "../..//Algorithms/Motor_Control/Motor_Control.h"
#include "Swarming.h"

```

Include dependency graph for Swarming.c:



Macros

- `#define Photodiode_Ratio_Maximum 1.2`
The maximum ratio value of photodiodes to be considered "aligned."
- `#define Photodiode_Ratio_Minimum 0.8`
The minimum ratio value of photodiodes to be considered "aligned."

Enumerations

- `enum Region_Value {`
`Red_Region_Value, Green_Region_Value, Yellow_Region_Value, Outside_Region_Value,`
`Front_Region_Value, Center_Region_Value, Rear_Region_Value }`
The value associated with the various regions.

Functions

- `uint8_t Swarming_Region` (int Current_Distance)
- `uint8_t Swarming_Range_Left_Right` (float Ratio)
- `void Swarming_Right_Calculation` (int *X, int *Y, int Photodiode_Value_Front, int Photodiode_Value_Rear, float Ratio)
- `void Swarming_Left_Calculation` (int *X, int *Y, int Photodiode_Value_Front, int Photodiode_Value_Rear, float Ratio)
- `void Swarming_Forward_Calculation` (int *X, int *Y, int Photodiode_Value)
- `int Swarming_Retrieve_X_Offset` (void)

- int [Swarming_Retrieve_Y_Offset](#) (void)
- void [Swarming](#) (void)
- void [Swarming_Update](#) (void)
- void [Swarming_Enable_Flag](#) (void)

3.5.1 Detailed Description

This file is code for the swarming algorithm.

Author

Ryan Lipski, Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 4/29/2015

Created: 2/12/2015 11:35:12 PM

Definition in file [Swarming.c](#).

3.5.2 Macro Definition Documentation

3.5.2.1 #define Photodiode_Ratio_Maximum 1.2

The maximum ratio value of photodiodes to be considered "aligned".

Definition at line 10 of file [Swarming.c](#).

Referenced by [Swarming_Range_Left_Right\(\)](#).

3.5.2.2 #define Photodiode_Ratio_Minimum 0.8

The minimum ratio value of photodiodes to be considered "aligned".

Definition at line 12 of file [Swarming.c](#).

Referenced by [Swarming_Range_Left_Right\(\)](#).

3.5.3 Enumeration Type Documentation

3.5.3.1 enum Region_Value

The value associated with the various regions.

Enumerator

Red_Region_Value
Green_Region_Value
Yellow_Region_Value
Outside_Region_Value
Front_Region_Value
Center_Region_Value
Rear_Region_Value

Definition at line 15 of file [Swarming.c](#).

3.5.4 Function Documentation

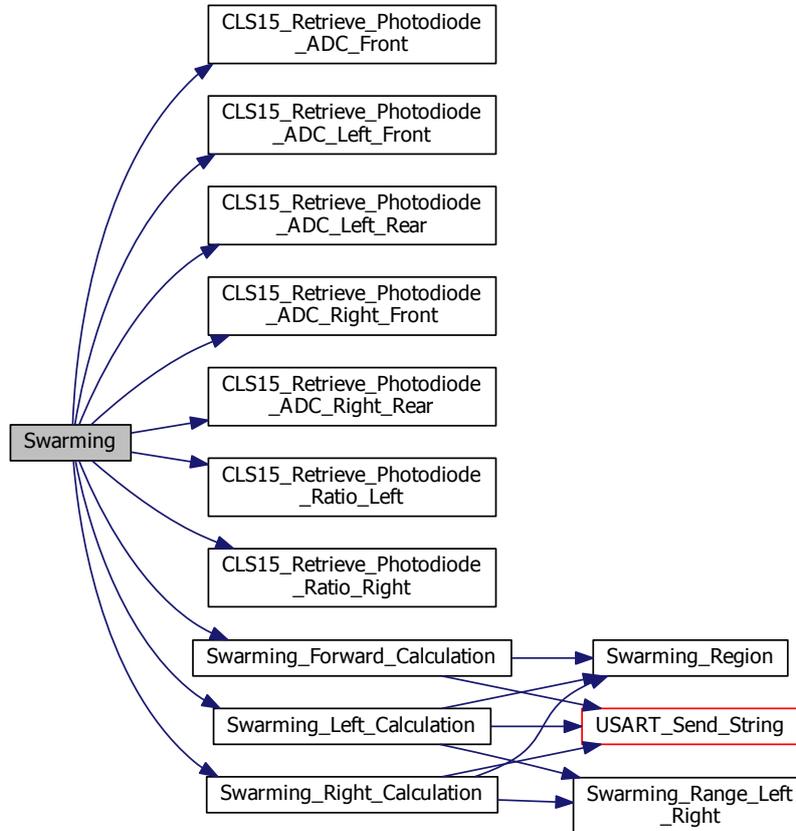
3.5.4.1 void Swarming (void)

Definition at line 343 of file Swarming.c.

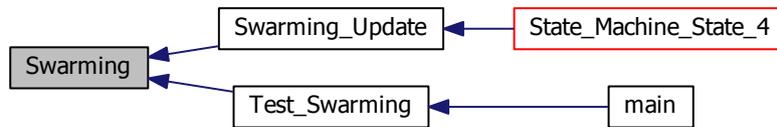
References [CLS15_Retrieve_Photodiode_ADC_Front\(\)](#), [CLS15_Retrieve_Photodiode_ADC_Left_Front\(\)](#), [CLS15_Retrieve_Photodiode_ADC_Left_Rear\(\)](#), [CLS15_Retrieve_Photodiode_ADC_Right_Front\(\)](#), [CLS15_Retrieve_Photodiode_ADC_Right_Rear\(\)](#), [CLS15_Retrieve_Photodiode_Ratio_Left\(\)](#), [CLS15_Retrieve_Photodiode_Ratio_Right\(\)](#), [Swarming_Forward_Calculation\(\)](#), [Swarming_Left_Calculation\(\)](#), and [Swarming_Right_Calculation\(\)](#).

Referenced by [Swarming_Update\(\)](#), and [Test_Swarming\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.5.4.2 void Swarming_Enable_Flag (void)

Definition at line 364 of file [Swarming.c](#).

Referenced by [ISR\(\)](#).

Here is the caller graph for this function:



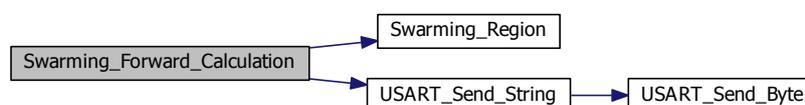
3.5.4.3 void Swarming_Forward_Calculation (int * X, int * Y, int Photodiode_Value) [inline]

Definition at line 297 of file [Swarming.c](#).

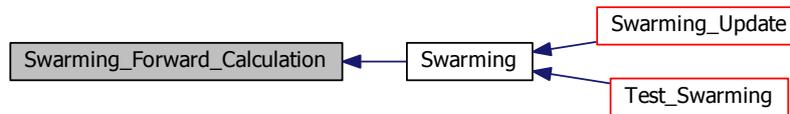
References [Green_Region_Value](#), [Red_Region_Value](#), [Swarming_Region\(\)](#), [USART_Send_String\(\)](#), and [Yellow_Region_Value](#).

Referenced by [Swarming\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



```

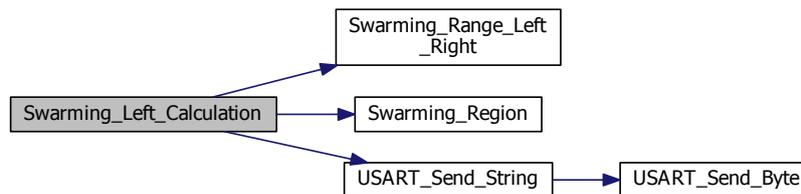
3.5.4.4 void Swarming_Left_Calculation ( int * X, int * Y, int Photodiode_Value_Front, int Photodiode_Value_Rear, float Ratio
) [inline]
  
```

Definition at line 188 of file [Swarming.c](#).

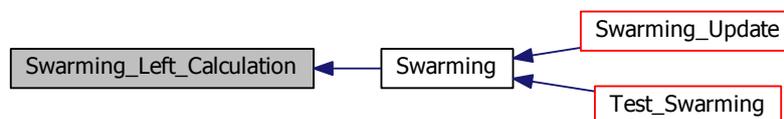
References [Center_Region_Value](#), [Front_Region_Value](#), [Green_Region_Value](#), [Outside_Region_Value](#), [Rear_Region_Value](#), [Red_Region_Value](#), [Swarming_Range_Left_Right\(\)](#), [Swarming_Region\(\)](#), [USART_Send_String\(\)](#), and [Yellow_Region_Value](#).

Referenced by [Swarming\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



```

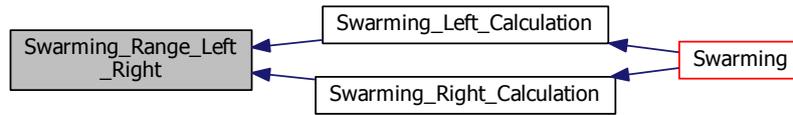
3.5.4.5 uint8_t Swarming_Range_Left_Right ( float Ratio ) [inline]
  
```

Definition at line 64 of file [Swarming.c](#).

References [Center_Region_Value](#), [Front_Region_Value](#), [Photodiode_Ratio_Maximum](#), [Photodiode_Ratio_Minimum](#), and [Rear_Region_Value](#).

Referenced by [Swarming_Left_Calculation\(\)](#), and [Swarming_Right_Calculation\(\)](#).

Here is the caller graph for this function:



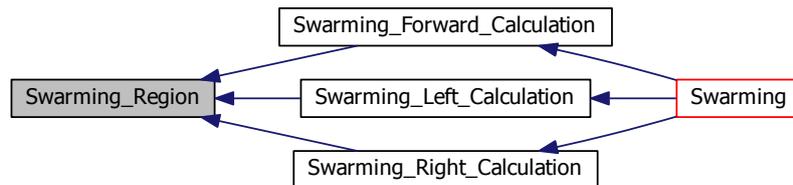
3.5.4.6 `uint8_t Swarming_Region (int Current_Distance) [inline]`

Definition at line 44 of file [Swarming.c](#).

References [Green_Region_Value](#), [Outside_Region_Value](#), [Radius_Green](#), [Radius_Red](#), [Radius_Yellow](#), [Red_Region_Value](#), and [Yellow_Region_Value](#).

Referenced by [Swarming_Forward_Calculation\(\)](#), [Swarming_Left_Calculation\(\)](#), and [Swarming_Right_Calculation\(\)](#).

Here is the caller graph for this function:

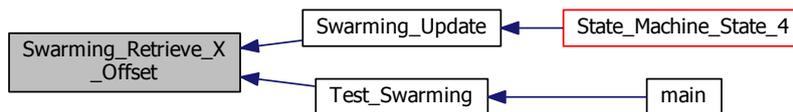


3.5.4.7 `int Swarming_Retrieve_X_Offset (void)`

Definition at line 333 of file [Swarming.c](#).

Referenced by [Swarming_Update\(\)](#), and [Test_Swarming\(\)](#).

Here is the caller graph for this function:

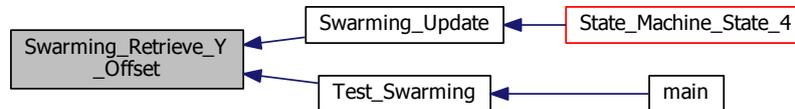


3.5.4.8 `int Swarming_Retrieve_Y_Offset (void)`

Definition at line 338 of file [Swarming.c](#).

Referenced by [Swarming_Update\(\)](#), and [Test_Swarming\(\)](#).

Here is the caller graph for this function:



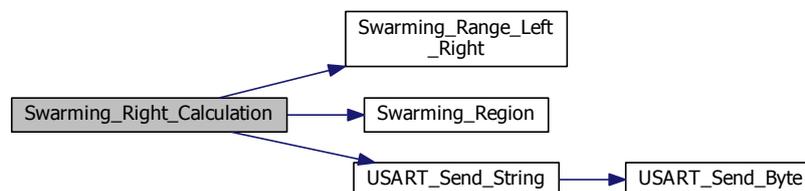
3.5.4.9 `void Swarming_Right_Calculation (int * X, int * Y, int Photodiode_Value_Front, int Photodiode_Value_Rear, float Ratio) [inline]`

Definition at line 80 of file [Swarming.c](#).

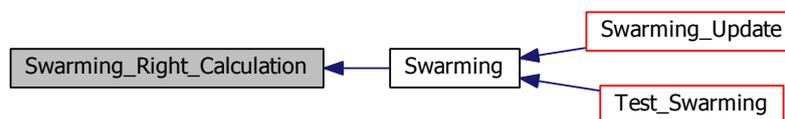
References [Center_Region_Value](#), [Front_Region_Value](#), [Green_Region_Value](#), [Outside_Region_Value](#), [Rear_Region_Value](#), [Red_Region_Value](#), [Swarming_Range_Left_Right\(\)](#), [Swarming_Region\(\)](#), [USART_Send_String\(\)](#), and [Yellow_Region_Value](#).

Referenced by [Swarming\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



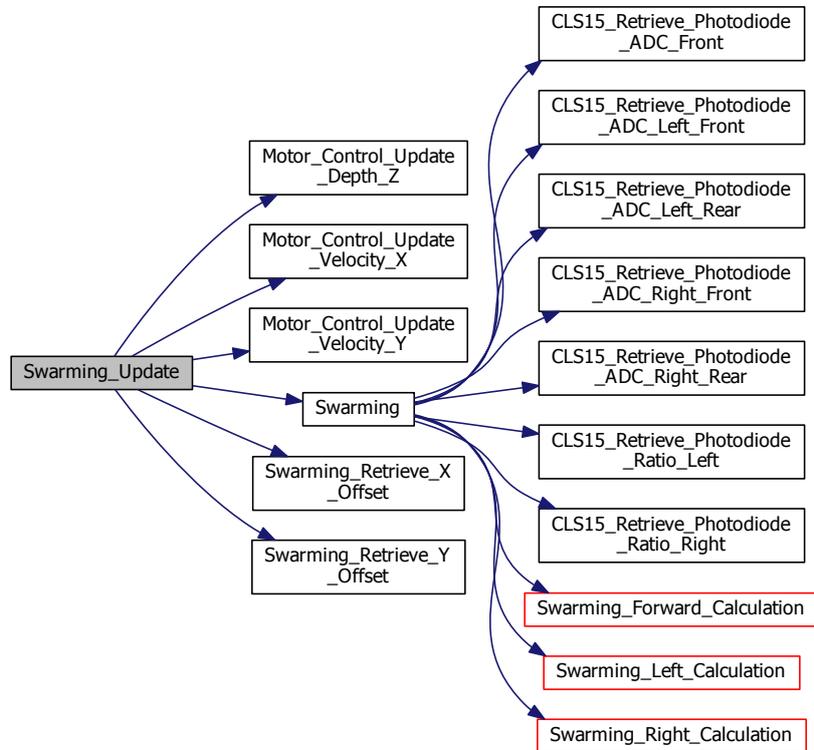
3.5.4.10 `void Swarming_Update (void)`

Definition at line 352 of file [Swarming.c](#).

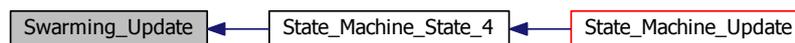
References [Motor_Control_Update_Depth_Z\(\)](#), [Motor_Control_Update_Velocity_X\(\)](#), [Motor_Control_Update_Velocity_Y\(\)](#), [Swarming\(\)](#), [Swarming_Retrieve_X_Offset\(\)](#), and [Swarming_Retrieve_Y_Offset\(\)](#).

Referenced by [State_Machine_State_4\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.6 Swarming.c

```

00001 /**
00002  * @file Swarming.c
00003  * @author Ryan Lipski, Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 4/29/2015\n Created: 2/12/2015 11:35:12 PM
00006  * @brief This file is code for the swarming algorithm.
00007  */
00008
00009 /** @brief The maximum ratio value of photodiodes to be considered "aligned." */
00010 #define Photodiode_Ratio_Maximum 1.2
00011 /** @brief The minimum ratio value of photodiodes to be considered "aligned." */
00012 #define Photodiode_Ratio_Minimum 0.8
00013
00014 /** @brief The value associated with the various regions. */
00015 enum Region_Value
00016 {
00017     Red_Region_Value,
00018     Green_Region_Value,
  
```

```

00019     Yellow_Region_Value,
00020     Outside_Region_Value,
00021     Front_Region_Value,
00022     Center_Region_Value,
00023     Rear_Region_Value,
00024 };
00025
00026 #include <stdint.h>
00027 #include "../Master_Build_Control.h"
00028 #include "../Subsystem/Light/CLS15.h"
00029 #include "../Algorithms/Motor_Control/Motor_Control.h"
00030 #include "Swarming.h"
00031
00032 #if Print_Region == 1
00033     #include <stdio.h>
00034     #include "../Microcontroller/Communication/USART.h"
00035 #endif
00036
00037 /* X offset from the swarming code. */
00038 static int X_Offset = 0;
00039 /* Y offset from the swarming code. */
00040 static int Y_Offset = 0;
00041 /* Flag to signify that the swarming algorithm is to be updated. */
00042 static int Update_Swarming_Flag = 1;
00043
00044 inline uint8_t Swarming_Region(int Current_Distance)
00045 {
00046     if(Current_Distance>=Radius_Red)
00047     {
00048         return(Red_Region_Value);
00049     }
00050     else if(Current_Distance>=Radius_Green)
00051     {
00052         return(Green_Region_Value);
00053     }
00054     else if(Current_Distance>=Radius_Yellow)
00055     {
00056         return(Yellow_Region_Value);
00057     }
00058     else
00059     {
00060         return(Outside_Region_Value);
00061     }
00062 }
00063
00064 inline uint8_t Swarming_Range_Left_Right(float Ratio)
00065 {
00066     if(Ratio>Photodiode_Ratio_Maximum)
00067     {
00068         return(Front_Region_Value);
00069     }
00070     else if(Ratio<Photodiode_Ratio_Minimum)
00071     {
00072         return(Rear_Region_Value);
00073     }
00074     else
00075     {
00076         return(Center_Region_Value);
00077     }
00078 }
00079
00080 inline void Swarming_Right_Calculation(int *X, int *Y, int
Photodiode_Value_Front, int Photodiode_Value_Rear, float Ratio)
00081 {
00082     uint8_t Ratio_Region = Swarming_Range_Left_Right(Ratio);
00083     uint8_t Front_Region = Swarming_Region(Photodiode_Value_Front);
00084     uint8_t Rear_Region = Swarming_Region(Photodiode_Value_Rear);
00085     uint8_t Check_Region = Outside_Region_Value;
00086     #if Print_Region == 1
00087         char Buffer[20] = {0x00};
00088         sprintf(Buffer, "Left %d, %d", Photodiode_Value_Front, Photodiode_Value_Rear);
00089         USART_Send_String(Buffer);
00090     #endif
00091     if(Front_Region <= Rear_Region)
00092     {
00093         Check_Region = Front_Region;
00094     }
00095     else
00096     {
00097         Check_Region = Rear_Region;
00098     }
00099     switch(Check_Region)
00100     {
00101         case Red_Region_Value: //check the near active region
00102             switch(Ratio_Region)
00103             {
00104                 case Front_Region_Value:

```

```

00105         //turn left b*Y a percentage and decrease forward speed b*Y a percentage
00106         *X+=2;//left = +, right = -
00107         #if Print_Region == 1
00108             USART_Send_String("Right front red region\n\r");
00109         #endif
00110         break;
00111     case Center_Region_Value:
00112         //turn left b*Y a percentage
00113         *X+=1;//left = +, right = -
00114         #if Print_Region == 1
00115             USART_Send_String("Right center red region\n\r");
00116         #endif
00117         break;
00118     case Rear_Region_Value:
00119         //turn left b*Y a percentage and increase forward speed b*Y a percentage
00120         *X+=1;//left = +, right = -
00121         *Y-=1;//forward = +, reverse = -
00122         #if Print_Region == 1
00123             USART_Send_String("Right rear red region\n\r");
00124         #endif
00125         break;
00126     }
00127     break;
00128 case Green_Region_Value:
00129     switch(Ratio_Region)
00130     {
00131     case Front_Region_Value:
00132         //increase forward speed b*Y a percentage
00133         *Y+=1;//forward = +, reverse = -
00134         #if Print_Region == 1
00135             USART_Send_String("Right front green region\n\r");
00136         #endif
00137         break;
00138     case Center_Region_Value:
00139         #if Print_Region == 1
00140             USART_Send_String("Right center green region\n\r");
00141         #endif
00142         break;
00143     case Rear_Region_Value:
00144         //decrease forward speed b*Y a percentage
00145         *Y-=1;//forward = +, reverse = -
00146         #if Print_Region == 1
00147             USART_Send_String("Right rear green region\n\r");
00148         #endif
00149         break;
00150     }
00151     break;
00152 case Yellow_Region_Value:
00153     switch(Ratio_Region)
00154     {
00155     case Front_Region_Value:
00156         //turn right b*Y a percentage and decrease forward speed b*Y a percentage
00157         *X+=1;//left = +, right = -
00158         *Y+=1;//forward = +, reverse = -
00159         #if Print_Region == 1
00160             USART_Send_String("Right front yellow region\n\r");
00161         #endif
00162         break;
00163     case Center_Region_Value:
00164         //turn right b*Y a percentage
00165         *X+=1;//left = +, right = -
00166         #if Print_Region == 1
00167             USART_Send_String("Right center yellow region\n\r");
00168         #endif
00169         break;
00170     case Rear_Region_Value:
00171         //turn right b*Y a percentage and increase forward speed b*Y a percentage
00172         *X+=1;//left = +, right = -
00173         *Y-=1;//forward = +, reverse = -
00174         #if Print_Region == 1
00175             USART_Send_String("Right rear yellow region\n\r");
00176         #endif
00177         break;
00178     }
00179     break;
00180 default:
00181     #if Print_Region == 1
00182         USART_Send_String("Right outside region\n\r");
00183     #endif
00184     break;
00185 }
00186 }
00187
00188 inline void Swarming_Left_Calculation(int *X, int *Y, int Photodiode_Value_Front
, int Photodiode_Value_Rear, float Ratio)
00189 {
00190     uint8_t Ratio_Region = Swarming_Range_Left_Right (Ratio);

```

```

00191     uint8_t Front_Region = Swarming_Region(Photodiode_Value_Front);
00192     uint8_t Rear_Region = Swarming_Region(Photodiode_Value_Rear);
00193     uint8_t Check_Region = Outside_Region_Value;
00194     #if Print_Region == 1
00195         char Buffer[20] = {0x00};
00196         sprintf(Buffer, "Left %d, %d", Photodiode_Value_Front, Photodiode_Value_Rear);
00197         USART_Send_String(Buffer);
00198     #endif
00199     if(Front_Region <= Rear_Region)
00200     {
00201         Check_Region = Front_Region;
00202     }
00203     else
00204     {
00205         Check_Region = Rear_Region;
00206     }
00207     switch(Check_Region)
00208     {
00209         case Red_Region_Value: //check the near active region
00210             switch(Ratio_Region)
00211             {
00212                 case Front_Region_Value:
00213                     //turn left b*Y a percentage and decrease forward speed b*Y a percentage
00214                     *X-=2;//left = +, right = -
00215                     *Y-=1;//forward = +, reverse = -
00216                     #if Print_Region == 1
00217                         USART_Send_String("Left front red region\n\r");
00218                     #endif
00219                     break;
00220                 case Center_Region_Value:
00221                     //turn left b*Y a percentage
00222                     *X-=1;//left = +, right = -
00223                     #if Print_Region == 1
00224                         USART_Send_String("Left center red region\n\r");
00225                     #endif
00226                     break;
00227                 case Rear_Region_Value:
00228                     //turn left b*Y a percentage and increase forward speed b*Y a percentage
00229                     *X-=1;//left = +, right = -
00230                     *Y+=1;//forward = +, reverse = -
00231                     #if Print_Region == 1
00232                         USART_Send_String("Left rear red region\n\r");
00233                     #endif
00234                     break;
00235             }
00236             break;
00237         case Green_Region_Value:
00238             switch(Ratio_Region)
00239             {
00240                 case Front_Region_Value:
00241                     //increase forward speed b*Y a percentage
00242                     *Y+=1;//forward = +, reverse = -
00243                     #if Print_Region == 1
00244                         USART_Send_String("Left front green region\n\r");
00245                     #endif
00246                     break;
00247                 case Center_Region_Value:
00248                     #if Print_Region == 1
00249                         USART_Send_String("Left center green region\n\r");
00250                     #endif
00251                     break;
00252                 case Rear_Region_Value:
00253                     //decrease forward speed b*Y a percentage
00254                     *Y-=1;//forward = +, reverse = -
00255                     #if Print_Region == 1
00256                         USART_Send_String("Left rear green region\n\r");
00257                     #endif
00258                     break;
00259             }
00260             break;
00261         case Yellow_Region_Value:
00262             switch(Ratio_Region)
00263             {
00264                 case Front_Region_Value:
00265                     //turn right b*Y a percentage and decrease forward speed b*Y a percentage
00266                     *X-=1;//left = +, right = -
00267                     *Y+=1;//forward = +, reverse = -
00268                     #if Print_Region == 1
00269                         USART_Send_String("Left front yellow region\n\r");
00270                     #endif
00271                     break;
00272                 case Center_Region_Value:
00273                     //turn right b*Y a percentage
00274                     *X-=1;//left = +, right = -
00275                     #if Print_Region == 1
00276                         USART_Send_String("Left center yellow region\n\r");
00277                     #endif

```

```

00278         break;
00279     case Rear_Region_Value:
00280         //turn right b*Y a percentage and increase forward speed b*Y a percentage
00281         *X-=1;//left = +, right = -
00282         *Y-=1;//forward = +, reverse = -
00283         #if Print_Region == 1
00284             USART_Send_String("Left rear yellow region\n\r");
00285         #endif
00286         break;
00287     }
00288     break;
00289 default:
00290     #if Print_Region == 1
00291         USART_Send_String("Left outside region\n\r");
00292     #endif
00293     break;
00294 }
00295 }
00296
00297 inline void Swarming_Forward_Calculation(int *X, int *
Y, int Photodiode_Value)
00298 {
00299     #if Print_Region == 1
00300     char Buffer[20] = {0x00};
00301     sprintf(Buffer, "Front %d", Photodiode_Value);
00302     USART_Send_String(Buffer);
00303     #endif
00304     switch(Swarming_Region(Photodiode_Value))
00305     {
00306     case Red_Region_Value: //check the near active region
00307         //turn left b*Y a percentage and reduce forward speed b*Y a percentage
00308         *X=5;//left = +, right = -
00309         *Y=0;//forward = +, reverse = -
00310         #if Print_Region == 1
00311             USART_Send_String("Front red region\n\r");
00312         #endif
00313         break;
00314     case Green_Region_Value: //check the far active region
00315         //turn left b*Y a percentage and reduce forward speed b*Y a percentage
00316         *X=5;//left = +, right = -
00317         *Y=0;//forward = +, reverse = -
00318         #if Print_Region == 1
00319             USART_Send_String("Front yellow region\n\r");
00320         #endif
00321         break;
00322     case Yellow_Region_Value: //check the far active region
00323         //turn left b*Y a percentage and reduce forward speed b*Y a percentage
00324         *X=5;//left = +, right = -
00325         *Y=0;//forward = +, reverse = -
00326         #if Print_Region == 1
00327             USART_Send_String("Front yellow region\n\r");
00328         #endif
00329         break;
00330     }
00331 }
00332
00333 int Swarming_Retrieve_X_Offset(void)
00334 {
00335     return(X_Offset);
00336 }
00337
00338 int Swarming_Retrieve_Y_Offset(void)
00339 {
00340     return(Y_Offset);
00341 }
00342
00343 void Swarming(void)
00344 {
00345     X_Offset = 0; // *X-plane motor control variable, left+= and right=- movements
00346     Y_Offset = 3; // *Y-plane motor control variable, forward+= and reverse=- movements
00347     Swarming_Left_Calculation(&X_Offset, &Y_Offset,
CLS15_Retrieve_Photodiode_ADC_Left_Front(),
CLS15_Retrieve_Photodiode_ADC_Left_Rear(),
CLS15_Retrieve_Photodiode_Ratio_Left());
00348     Swarming_Right_Calculation(&X_Offset, &Y_Offset,
CLS15_Retrieve_Photodiode_ADC_Right_Front(),
CLS15_Retrieve_Photodiode_ADC_Right_Rear(),
CLS15_Retrieve_Photodiode_Ratio_Right());
00349     Swarming_Forward_Calculation(&X_Offset, &Y_Offset,
CLS15_Retrieve_Photodiode_ADC_Front());
00350 }
00351
00352 void Swarming_Update(void)
00353 {
00354     if(Update_Swarming_Flag)
00355     {
00356         Swarming();

```

```

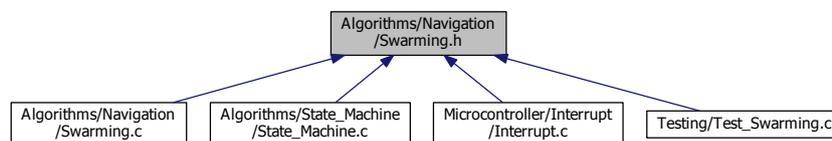
00357     Motor_Control_Update_Velocity_X(
Swarming_Retrieve_X_Offset());
00358     Motor_Control_Update_Velocity_Y(
Swarming_Retrieve_Y_Offset());
00359     Motor_Control_Update_Depth_Z(2.0);
00360     Update_Swarming_Flag = 0;
00361 }
00362 }
00363
00364 void Swarming_Enable_Flag(void)
00365 {
00366     Update_Swarming_Flag = 1;
00367 }

```

3.7 Algorithms/Navigation/Swarming.h File Reference

This file is the header file for the swarming algorithm.

This graph shows which files directly or indirectly include this file:



Enumerations

- enum [Region_Radius](#) { [Radius_Red](#) = 750, [Radius_Green](#) = 400, [Radius_Yellow](#) = 100 }
Maximum radius for each of the ranges in inches.

Functions

- void [Swarming](#) (void)
- int [Swarming_Retrieve_X_Offset](#) (void)
- int [Swarming_Retrieve_Y_Offset](#) (void)
- void [Swarming_Update](#) (void)
- void [Swarming_Enable_Flag](#) (void)

3.7.1 Detailed Description

This file is the header file for the swarming algorithm.

Author

Ryan Lipski, Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 4/29/2015
Created: 2/12/2015 11:35:36 PM

Definition in file [Swarming.h](#).

3.7.2 Enumeration Type Documentation

3.7.2.1 enum Region_Radius

Maximum radius for each of the ranges in inches.

Enumerator

Radius_Red

Radius_Green

Radius_Yellow

Definition at line 13 of file [Swarming.h](#).

3.7.3 Function Documentation

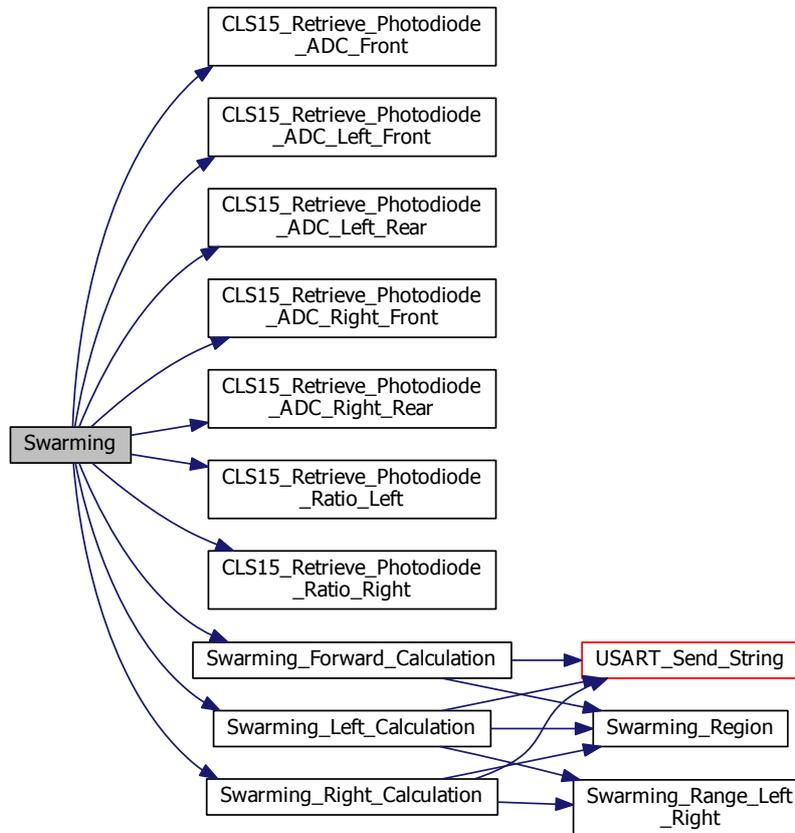
3.7.3.1 void Swarming (void)

Definition at line 343 of file [Swarming.c](#).

References [CLS15_Retrieve_Photodiode_ADC_Front\(\)](#), [CLS15_Retrieve_Photodiode_ADC_Left_Front\(\)](#), [CLS15_Retrieve_Photodiode_ADC_Left_Rear\(\)](#), [CLS15_Retrieve_Photodiode_ADC_Right_Front\(\)](#), [CLS15_Retrieve_Photodiode_ADC_Right_Rear\(\)](#), [CLS15_Retrieve_Photodiode_Ratio_Left\(\)](#), [CLS15_Retrieve_Photodiode_Ratio_Right\(\)](#), [Swarming_Forward_Calculation\(\)](#), [Swarming_Left_Calculation\(\)](#), and [Swarming_Right_Calculation\(\)](#).

Referenced by [Swarming_Update\(\)](#), and [Test_Swarming\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.7.3.2 void Swarming_Enable_Flag (void)

Definition at line 364 of file [Swarming.c](#).

Referenced by [ISR\(\)](#).

Here is the caller graph for this function:

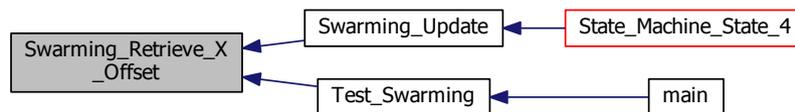


3.7.3.3 `int Swarming_Retrieve_X_Offset (void)`

Definition at line 333 of file [Swarming.c](#).

Referenced by [Swarming_Update\(\)](#), and [Test_Swarming\(\)](#).

Here is the caller graph for this function:

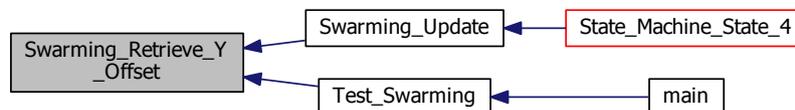


3.7.3.4 `int Swarming_Retrieve_Y_Offset (void)`

Definition at line 338 of file [Swarming.c](#).

Referenced by [Swarming_Update\(\)](#), and [Test_Swarming\(\)](#).

Here is the caller graph for this function:



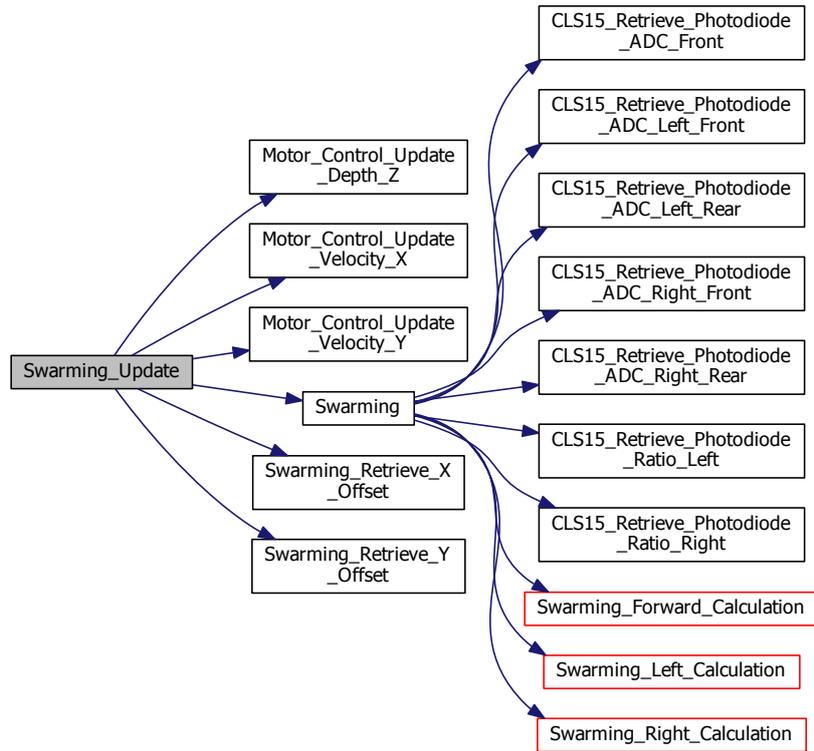
3.7.3.5 `void Swarming_Update (void)`

Definition at line 352 of file [Swarming.c](#).

References [Motor_Control_Update_Depth_Z\(\)](#), [Motor_Control_Update_Velocity_X\(\)](#), [Motor_Control_Update_Velocity_Y\(\)](#), [Swarming\(\)](#), [Swarming_Retrieve_X_Offset\(\)](#), and [Swarming_Retrieve_Y_Offset\(\)](#).

Referenced by [State_Machine_State_4\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.8 Swarming.h

```

00001 /**
00002  * @file Swarming.h
00003  * @author Ryan Lipski, Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 4/29/2015\n Created: 2/12/2015 11:35:36 PM
00006  * @brief This file is the header file for the swarming algorithm.
00007  */
00008
00009 #ifndef Swarming_H_
00010 #define Swarming_H_
00011
00012 /** @brief Maximum radius for each of the ranges in inches. */
00013 enum Region_Radius
00014 {
00015     Radius_Red = 750,
00016     Radius_Green = 400,
00017     Radius_Yellow = 100
00018 };
00019
00020 void Swarming(void);
  
```

```

00021 int Swarming_Retrieve_X_Offset(void);
00022 int Swarming_Retrieve_Y_Offset(void);
00023 void Swarming_Update(void);
00024 void Swarming_Enable_Flag(void);
00025
00026 #endif /* Swarming_H_ */

```

3.9 Algorithms/State_Machine/State_Machine.c File Reference

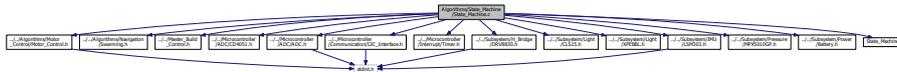
This file is code for the state machine code.

```

#include "../Algorithms/Motor_Control/Motor_Control.h"
#include "../Algorithms/Navigation/Swarming.h"
#include "../Master_Build_Control.h"
#include "../Microcontroller/ADC/CD4051.h"
#include "../Microcontroller/ADC/ADC.h"
#include "../Microcontroller/Communication/I2C_Interface.h"
#include "../Microcontroller/Interrupt/Timer.h"
#include "../Subsystem/H_Bridge/DRV8830.h"
#include "../Subsystem/Light/CLS15.h"
#include "../Subsystem/Light/XPEBBL.h"
#include "../Subsystem/IMU/LSM303.h"
#include "../Subsystem/Pressure/MPX5010GP.h"
#include "../Subsystem/Power/Battery.h"
#include "State_Machine.h"

```

Include dependency graph for State_Machine.c:



Functions

- void [State_Machine_Initialization](#) (void)
Transition routine for between state 0 and 1. Initialize all active subsystems.
- void [State_Machine_State_1](#) (void)
Routine for state 1. Indicate that the submarine is waiting to be depth.
- void [State_Machine_State_2](#) (void)
Routine for state 2. Update the heading for going forward.
- void [State_Machine_State_3](#) (void)
Routine for state 3. Calibrate the offsets for the photodiodes.
- void [State_Machine_State_4](#) (void)
Routine for state 4. Normal operation for the submarine.
- void [State_Machine_State_5](#) (void)
Routine for state 5. Toggling the front LED to indicate that .
- void [State_Machine_State_6](#) (void)
Routine for state 6. Indicate that the submarine is done.
- void [State_Machine_Transition_2](#) (void)
- void [State_Machine_Transition_3](#) (void)
- void [State_Machine_Transition_4](#) (void)
- void [State_Machine_Transition_5](#) (void)
- void [State_Machine_Transition_6](#) (void)
- void [State_Machine_Update](#) (void)
- void [State_Machine_Enable_Flag](#) (void)

3.9.1 Detailed Description

This file is code for the state machine code.

Author

Ryan Lipski, Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 4/29/2015

Created: 2/12/2015 2:13:56 PM

Definition in file [State_Machine.c](#).

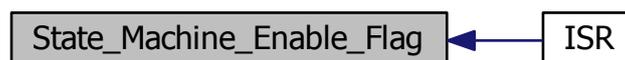
3.9.2 Function Documentation

3.9.2.1 void State_Machine_Enable_Flag (void)

Definition at line [392](#) of file [State_Machine.c](#).

Referenced by [ISR\(\)](#).

Here is the caller graph for this function:



3.9.2.2 void State_Machine_Initialization (void) [inline]

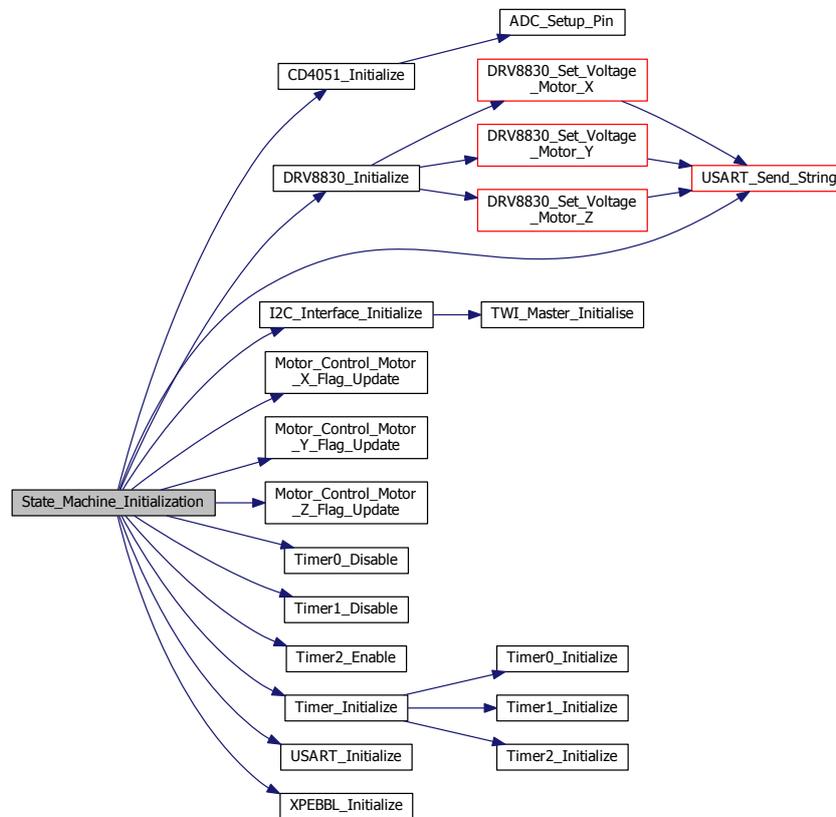
Transition routine for between state 0 and 1. Initialize all active subsystems.

Definition at line [32](#) of file [State_Machine.c](#).

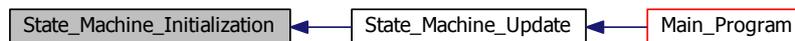
References [CD4051_Initialize\(\)](#), [DRV8830_Initialize\(\)](#), [I2C_Interface_Initialize\(\)](#), [Motor_Control_Motor_X_Flag_Update\(\)](#), [Motor_Control_Motor_Y_Flag_Update\(\)](#), [Motor_Control_Motor_Z_Flag_Update\(\)](#), [Timer0_Disable\(\)](#), [Timer1_Disable\(\)](#), [Timer2_Enable\(\)](#), [Timer_Initialize\(\)](#), [USART_Initialize\(\)](#), [USART_Send_String\(\)](#), and [XPEBBL_Initialize\(\)](#).

Referenced by [State_Machine_Update\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.9.2.3 void State_Machine_State_1 (void) [inline]

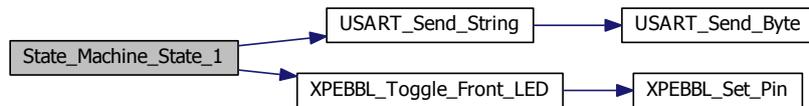
Routine for state 1. Indicate that the submarine is waiting to be depth.

Definition at line 65 of file [State_Machine.c](#).

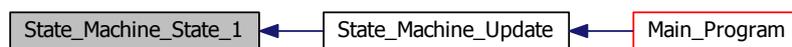
References [USART_Send_String\(\)](#), and [XPEBBL_Toggle_Front_LED\(\)](#).

Referenced by [State_Machine_Update\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.9.2.4 void State_Machine_State_2(void) [inline]

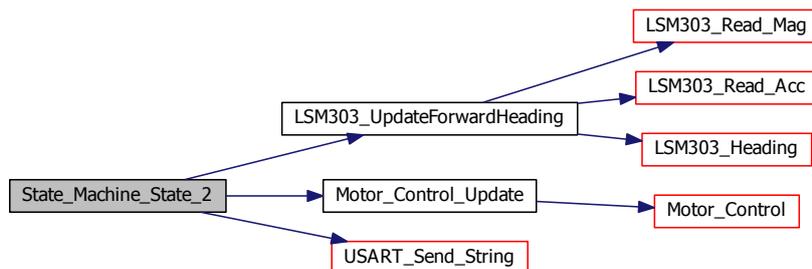
Routine for state 2. Update the heading for going forward.

Definition at line 75 of file [State_Machine.c](#).

References [LSM303_UpdateForwardHeading\(\)](#), [Motor_Control_Update\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [State_Machine_Update\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.9.2.5 void State_Machine_State_3 (void) [inline]

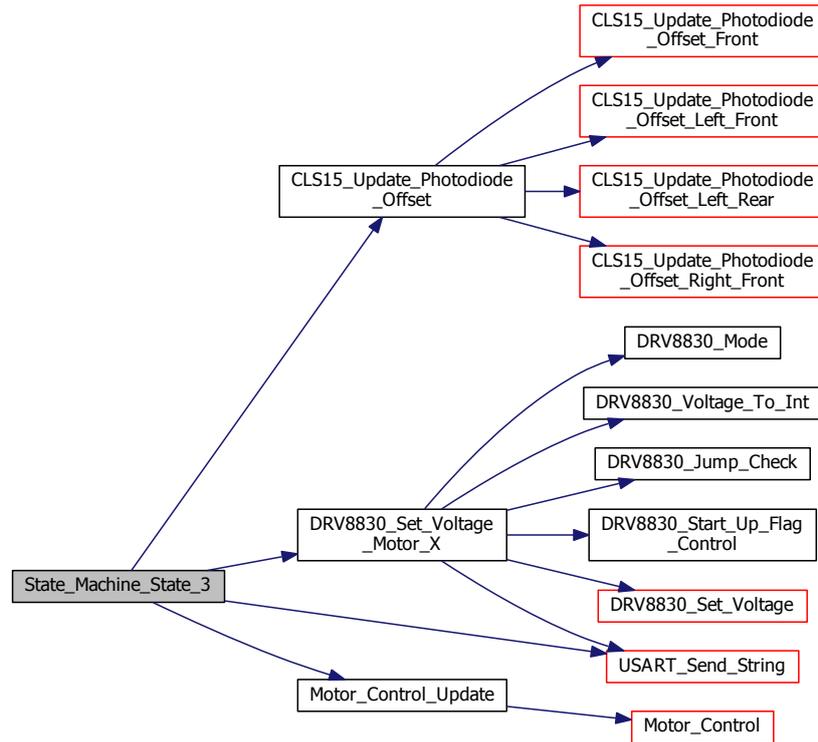
Routine for state 3. Calibrate the offsets for the photodiodes.

Definition at line 87 of file [State_Machine.c](#).

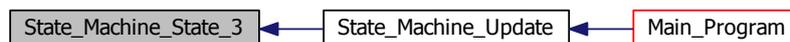
References [CLS15_Update_Photodiode_Offset\(\)](#), [DRV8830_Set_Voltage_Motor_X\(\)](#), [Motor_Control_Update\(\)](#), [Motor_Max_Voltage](#), and [USART_Send_String\(\)](#).

Referenced by [State_Machine_Update\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.9.2.6 void State_Machine_State_4 (void) [inline]

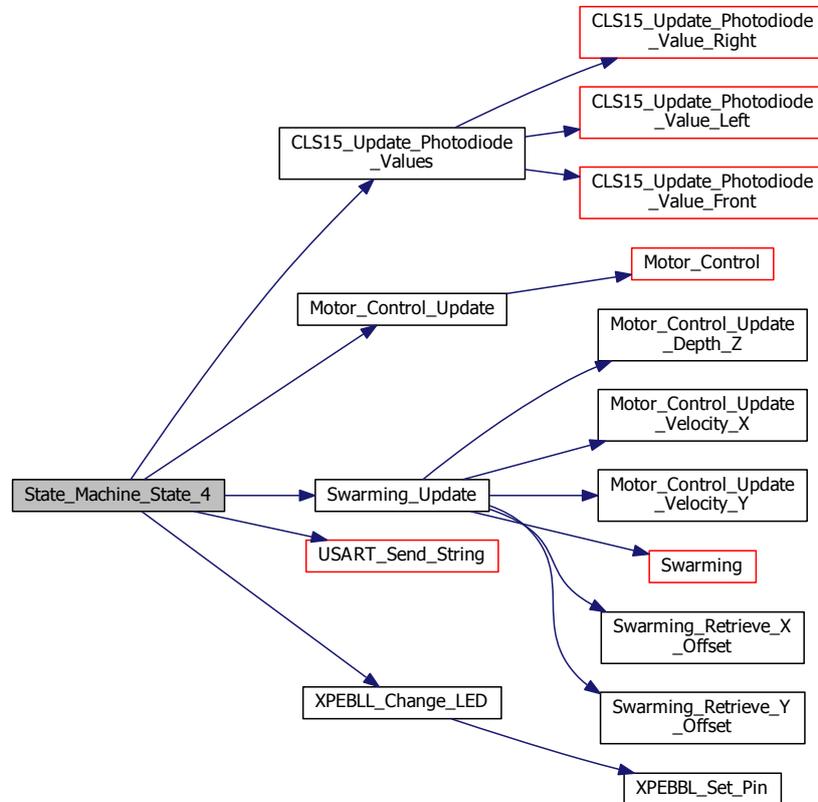
Routine for state 4. Normal operation for the submarine.

Definition at line 101 of file [State_Machine.c](#).

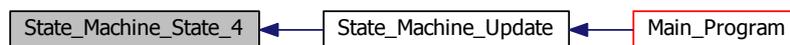
References [CLS15_Update_Photodiode_Values\(\)](#), [Motor_Control_Update\(\)](#), [Swarming_Update\(\)](#), [USART_Send_String\(\)](#), and [XPEBLL_Change_LED\(\)](#).

Referenced by [State_Machine_Update\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.9.2.7 void State_Machine_State_5 (void) [inline]

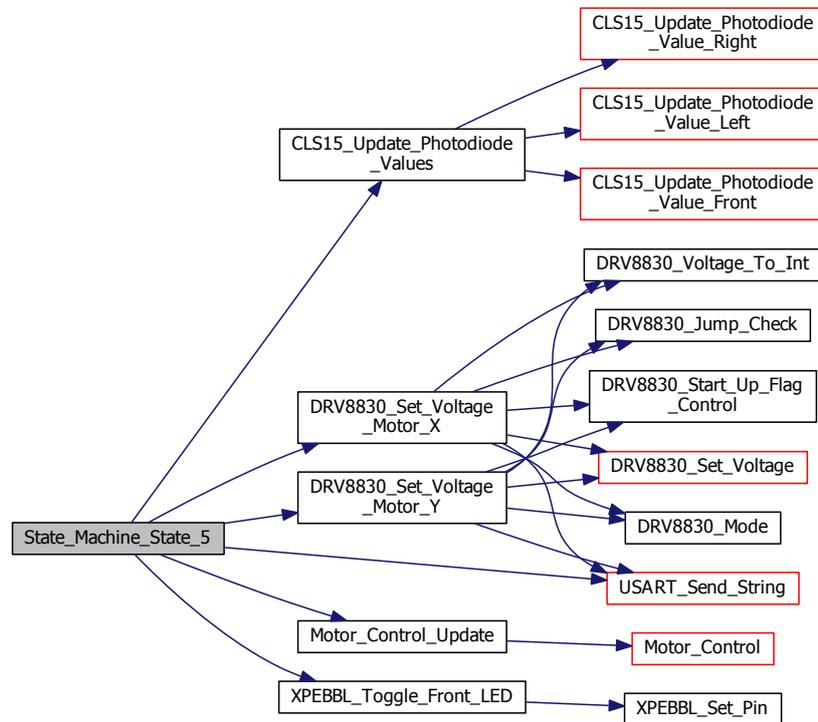
Routine for state 5. Toggling the front LED to indicate that .

Definition at line 115 of file [State_Machine.c](#).

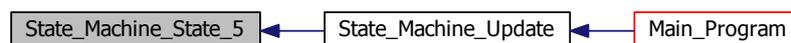
References [CLS15_Update_Phodiode_Values\(\)](#), [DRV8830_Set_Voltage_Motor_X\(\)](#), [DRV8830_Set_Voltage_Motor_Y\(\)](#), [Motor_Control_Update\(\)](#), [Motor_Max_Voltage](#), [USART_Send_String\(\)](#), and [XPEBLL_Toggle_Front_LED\(\)](#).

Referenced by [State_Machine_Update\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.9.2.8 void State_Machine_State_6 (void) [inline]

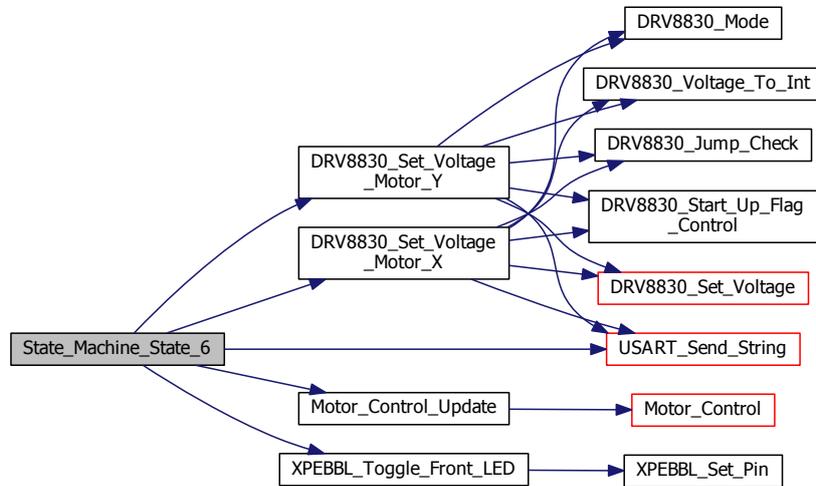
Routine for state 6. Indicate that the submarine is done.

Definition at line 131 of file [State_Machine.c](#).

References [DRV8830_Set_Voltage_Motor_X\(\)](#), [DRV8830_Set_Voltage_Motor_Y\(\)](#), [Motor_Control_Update\(\)](#), [USART_Send_String\(\)](#), and [XPEBBL_Toggle_Front_LED\(\)](#).

Referenced by [State_Machine_Update\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



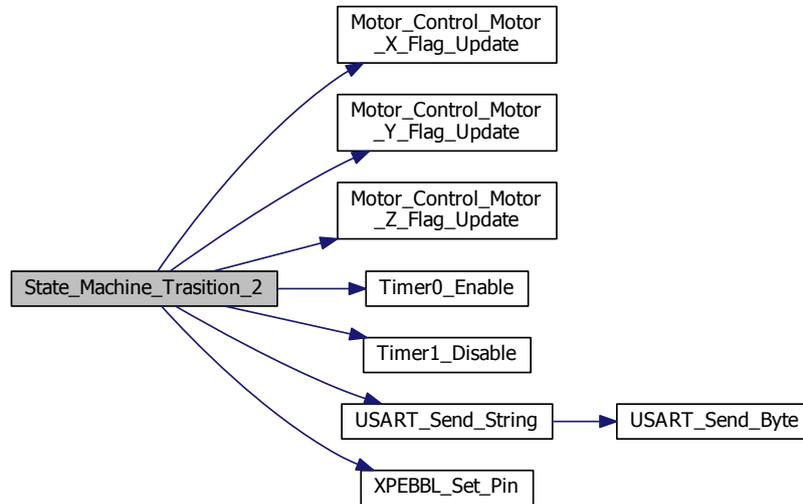
3.9.2.9 void State_Machine_Trasition_2 (void) [inline]

Definition at line 146 of file [State_Machine.c](#).

References [Motor_Control_Motor_X_Flag_Update\(\)](#), [Motor_Control_Motor_Y_Flag_Update\(\)](#), [Motor_Control_Motor_Z_Flag_Update\(\)](#), [Timer0_Enable\(\)](#), [Timer1_Disable\(\)](#), [USART_Send_String\(\)](#), and [XPEBBL_Set_Pin\(\)](#).

Referenced by [State_Machine_Update\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



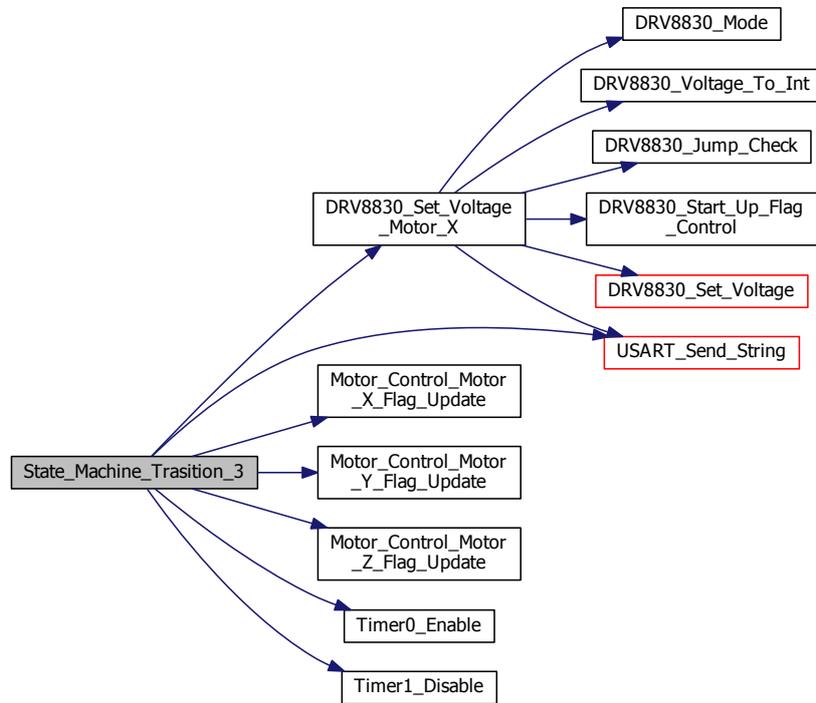
3.9.2.10 void State_Machine_Trasition_3 (void) [inline]

Definition at line 165 of file [State_Machine.c](#).

References [DRV8830_Set_Voltage_Motor_X\(\)](#), [Motor_Control_Motor_X_Flag_Update\(\)](#), [Motor_Control_Motor_Y_Flag_Update\(\)](#), [Motor_Control_Motor_Z_Flag_Update\(\)](#), [Motor_Max_Voltage](#), [Timer0_Enable\(\)](#), [Timer1_Disable\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [State_Machine_Update\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



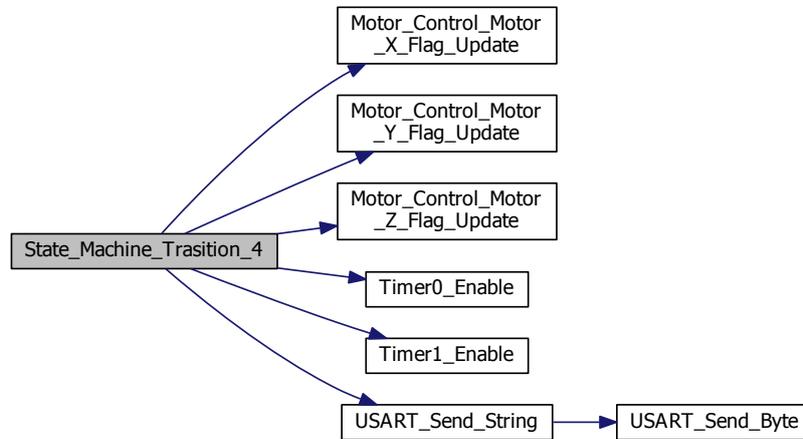
3.9.2.11 void State_Machine_Trasition_4 (void) [inline]

Definition at line 184 of file [State_Machine.c](#).

References [Motor_Control_Motor_X_Flag_Update\(\)](#), [Motor_Control_Motor_Y_Flag_Update\(\)](#), [Motor_Control_Motor_Z_Flag_Update\(\)](#), [Timer0_Enable\(\)](#), [Timer1_Enable\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [State_Machine_Update\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



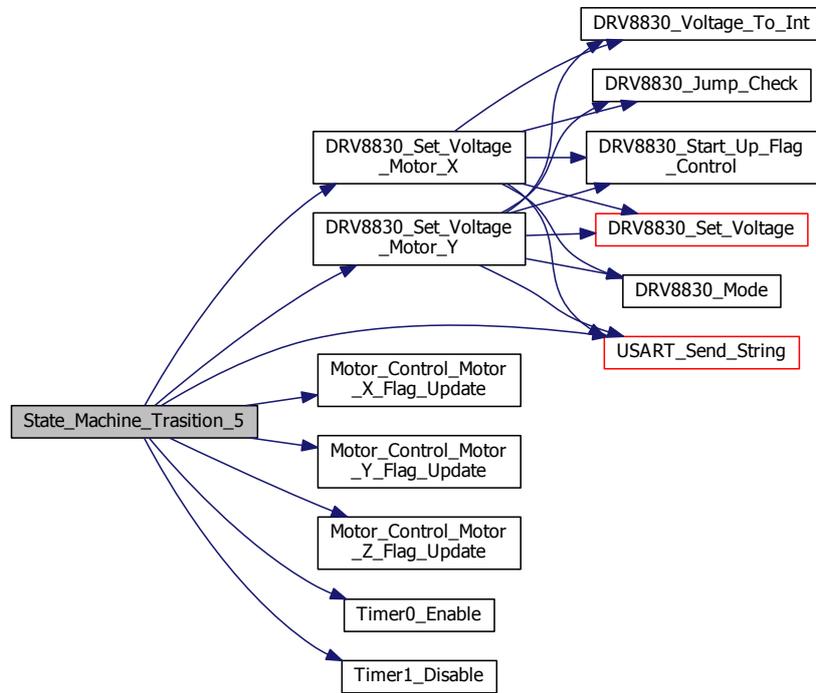
3.9.2.12 void State_Machine_Trasition_5 (void)

Definition at line 201 of file [State_Machine.c](#).

References [DRV8830_Set_Voltage_Motor_X\(\)](#), [DRV8830_Set_Voltage_Motor_Y\(\)](#), [Motor_Control_Motor_X_Flag_Update\(\)](#), [Motor_Control_Motor_Y_Flag_Update\(\)](#), [Motor_Control_Motor_Z_Flag_Update\(\)](#), [Motor_Max_Voltage](#), [Timer0_Enable\(\)](#), [Timer1_Disable\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [State_Machine_Update\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



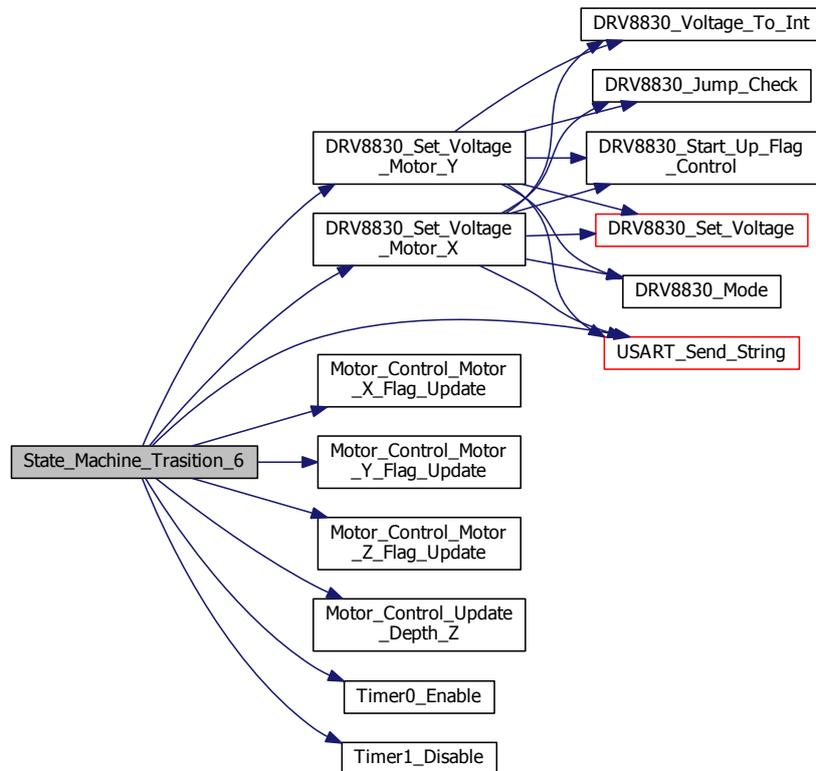
3.9.2.13 void State_Machine_Trasition_6 (void)

Definition at line 222 of file [State_Machine.c](#).

References [DRV8830_Set_Voltage_Motor_X\(\)](#), [DRV8830_Set_Voltage_Motor_Y\(\)](#), [Motor_Control_Motor_X_Flag_Update\(\)](#), [Motor_Control_Motor_Y_Flag_Update\(\)](#), [Motor_Control_Motor_Z_Flag_Update\(\)](#), [Motor_Control_Update_Depth_Z\(\)](#), [Timer0_Enable\(\)](#), [Timer1_Disable\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [State_Machine_Update\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



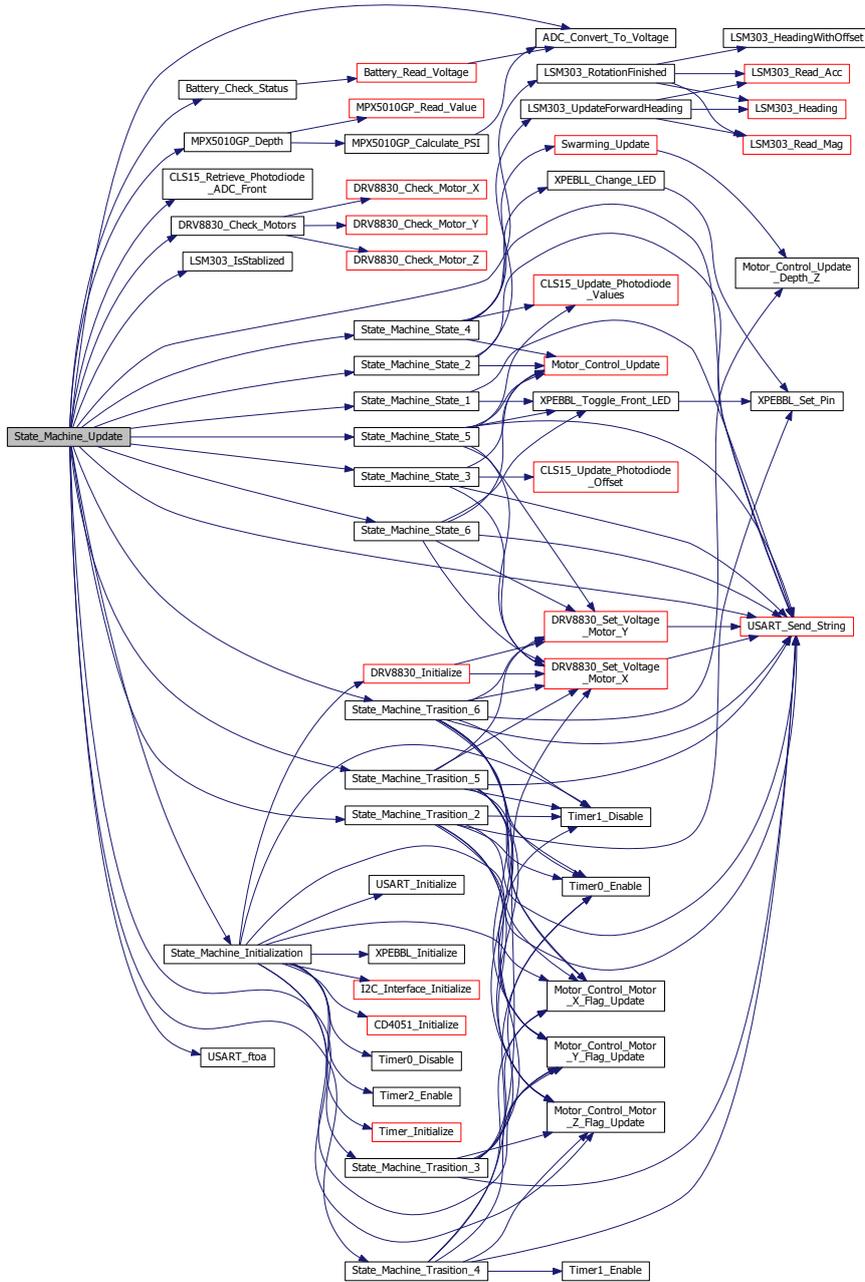
3.9.2.14 void State_Machine_Update (void)

Definition at line 245 of file [State_Machine.c](#).

References [ADC_Convert_To_Voltage\(\)](#), [Battery_Check_Status\(\)](#), [CLS15_Retrieve_Phodiode_ADC_Front\(\)](#), [DRV8830_Check_Motors\(\)](#), [FLOATING_POINT_BUFFER_SIZE](#), [LSM303_IsStablized\(\)](#), [LSM303_Rotation_Finished\(\)](#), [MPX5010GP_Depth\(\)](#), [Radius_Red](#), [State_Machine_Initialization\(\)](#), [State_Machine_State_1\(\)](#), [State_Machine_State_2\(\)](#), [State_Machine_State_3\(\)](#), [State_Machine_State_4\(\)](#), [State_Machine_State_5\(\)](#), [State_Machine_State_6\(\)](#), [State_Machine_Test_Mode](#), [State_Machine_Trasition_2\(\)](#), [State_Machine_Trasition_3\(\)](#), [State_Machine_Trasition_4\(\)](#), [State_Machine_Trasition_5\(\)](#), [State_Machine_Trasition_6\(\)](#), [USART_ftoa\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Main_Program\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.10 State_Machine.c

```

00001 /**
00002  * @file State_Machine.c
00003  * @author Ryan Lipski, Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 4/29/2015\n Created: 2/12/2015 2:13:56 PM
00006  * @brief This file is code for the state machine code.
00007  */
00008
00009 #include "../Algorithms/Motor_Control/Motor_Control.h"
00010 #include "../Algorithms/Navigation/Swarming.h"
00011 #include "../Master_Build_Control.h"
00012 #include "../Microcontroller/ADC/CD4051.h"
00013 #include "../Microcontroller/ADC/ADC.h"
00014 #include "../Microcontroller/Communication/I2C_Interface.h"
00015 #include "../Microcontroller/Interrupt/Timer.h"
00016 #include "../Subsystem/H_Bridge/DRV8830.h"
00017 #include "../Subsystem/Light/CLS15.h"
00018 #include "../Subsystem/Light/XPEBBL.h"
00019 #include "../Subsystem/IMU/LSM303.h"
00020 #include "../Subsystem/Pressure/MPX5010GP.h"
00021 #include "../Subsystem/Power/Battery.h"
00022 #include "State_Machine.h"
00023
00024 #if Print_State == 1 || Print_Region == 1
00025     #include <stdio.h>
00026     #include "../Microcontroller/Communication/USART.h"
00027 #endif
00028
00029 static int Update_State_Flag = 1;
00030
00031 /** @brief Transition routine for between state 0 and 1. Initialize all active subsystems. */
00032 inline void State_Machine_Initialization(void)
00033 {
00034     #if Print_State == 1 || Print_Region == 1
00035         USART_Initialize();
00036         USART_Send_String("State transition 1\n\r");
00037     #endif
00038     /* Initialize the TWI bus for the Compass and the H-bridge. */
00039     I2C_Interface_Initialize();
00040     /* Initialize the Compass subsystem. */
00041     //LSM303_Initialize();
00042     /* Initialize the h-bridges. */
00043     DRV8830_Initialize();
00044     /* Initialize the multiplexer. */
00045     CD4051_Initialize();
00046     /* Initialize the LEDs. */
00047     XPEBBL_Initialize();
00048     /* Initialize the timers. */
00049     Timer_Initialize();
00050     /* Disable the motor control for the X motor. */
00051     Motor_Control_Motor_X_Flag_Update(0);
00052     /* Disable the motor control for the Y motor. */
00053     Motor_Control_Motor_Y_Flag_Update(0);
00054     /* Disable the motor control for the Z motor. */
00055     Motor_Control_Motor_Z_Flag_Update(0);
00056     /* Disable Timer 0. */
00057     Timer0_Disable();
00058     /* Disable Timer 1. */
00059     Timer1_Disable();
00060     /* Enable Timer 2 to update the state machine. */
00061     Timer2_Enable();
00062 }
00063
00064 /** @brief Routine for state 1. Indicate that the submarine is waiting to be depth. */
00065 inline void State_Machine_State_1(void)
00066 {
00067     #if Print_State == 1
00068         USART_Send_String("State 1\n\r");
00069     #endif
00070     /* Indicate by toggling the front LED. */
00071     XPEBBL_Toggle_Front_LED();
00072 }
00073
00074 /** @brief Routine for state 2. Update the heading for going forward. */
00075 inline void State_Machine_State_2(void)
00076 {
00077     #if Print_State == 1
00078         USART_Send_String("State 2\n\r");
00079     #endif
00080     /* Update what is defined as the heading for going forward. */
00081     LSM303_UpdateForwardHeading();
00082     /* Update the motor control system. */
00083     Motor_Control_Update();
00084 }

```

```

00085
00086 /** @brief Routine for state 3. Calibrate the offsets for the photodiodes. */
00087 inline void State_Machine_State_3(void)
00088 {
00089     #if Print_State == 1
00090         USART_Send_String("State 3\n\r");
00091     #endif
00092     /* Set the turning to maximum. */
00093     DRV8830_Set_Voltage_Motor_X(Motor_Max_Voltage);
00094     /* Read the photodiodes and update the offset if needed. */
00095     CLS15_Update_Phodiode_Offset();
00096     /* Update the motor control system. */
00097     Motor_Control_Update();
00098 }
00099
00100 /** @brief Routine for state 4. Normal operation for the submarine. */
00101 inline void State_Machine_State_4(void)
00102 {
00103     #if Print_State == 1
00104         USART_Send_String("State 4\n\r");
00105     #endif
00106     /* Update the photodiodes taking into account the current active LED. */
00107     CLS15_Update_Phodiode_Values(
XPEBLL_Change_LED());
00108     /* Update the swarming values using the current LED values. */
00109     Swarming_Update();
00110     /* Update the motor control system. */
00111     Motor_Control_Update();
00112 }
00113
00114 /** @brief Routine for state 5. Toggling the front LED to indicate that . */
00115 inline void State_Machine_State_5(void)
00116 {
00117     #if Print_State == 1
00118         USART_Send_String("State 5\n\r");
00119     #endif
00120     /* Disable the turning motor. */
00121     DRV8830_Set_Voltage_Motor_X(0.0);
00122     /* Set the forward motor to maximum voltage. */
00123     DRV8830_Set_Voltage_Motor_Y(Motor_Max_Voltage);
00124     /* Update the front photodiodes taking into account the current active LED. */
00125     CLS15_Update_Phodiode_Values(
XPEBLL_Toggle_Front_LED()<<5);
00126     /* Update the motor control system. */
00127     Motor_Control_Update();
00128 }
00129
00130 /** @brief Routine for state 6. Indicate that the submarine is done. */
00131 inline void State_Machine_State_6(void)
00132 {
00133     #if Print_State == 1
00134         USART_Send_String("State 6\n\r");
00135     #endif
00136     /* Disable the turning motor. */
00137     DRV8830_Set_Voltage_Motor_X(0.0);
00138     /* Disable the forward motor. */
00139     DRV8830_Set_Voltage_Motor_Y(0.0);
00140     /* Indicate by toggling the front LED. */
00141     XPEBLL_Toggle_Front_LED();
00142     /* Update the motor control system. */
00143     Motor_Control_Update();
00144 }
00145
00146 inline void State_Machine_Trasition_2(void)
00147 {
00148     #if Print_State == 1
00149         USART_Send_String("State transition 2\n\r");
00150     #endif
00151     /* Turn off the LEDs. */
00152     XPEBLL_Set_Pin(0x00);
00153     /* Enable the motor control for the X motor. */
00154     Motor_Control_Motor_X_Flag_Update(0);
00155     /* Enable the motor control for the Y motor. */
00156     Motor_Control_Motor_Y_Flag_Update(0);
00157     /* Enable the motor control for the Z motor. */
00158     Motor_Control_Motor_Z_Flag_Update(1);
00159     /* Enable Timer 0 to update the motor control. */
00160     Timer0_Enable();
00161     /* Disable Timer 1. */
00162     Timer1_Disable();
00163 }
00164
00165 inline void State_Machine_Trasition_3(void)
00166 {
00167     #if Print_State == 1
00168         USART_Send_String("State transition 3\n\r");
00169     #endif

```

```

00170     /* Set the turning to maximum. */
00171     DRV8830_Set_Voltage_Motor_X(Motor_Max_Voltage);
00172     /* Enable the motor control for the X motor. */
00173     Motor_Control_Motor_X_Flag_Update(0);
00174     /* Enable the motor control for the Y motor. */
00175     Motor_Control_Motor_Y_Flag_Update(0);
00176     /* Enable the motor control for the Z motor. */
00177     Motor_Control_Motor_Z_Flag_Update(1);
00178     /* Enable Timer 0 to update the motor control. */
00179     Timer0_Enable();
00180     /* Disable Timer 1. */
00181     Timer1_Disable();
00182 }
00183
00184 inline void State_Machine_Trasition_4(void)
00185 {
00186     #if Print_State == 1
00187         USART_Send_String("State transition 4\n\r");
00188     #endif
00189     /* Enable the motor control for the X motor. */
00190     Motor_Control_Motor_X_Flag_Update(1);
00191     /* Enable the motor control for the Y motor. */
00192     Motor_Control_Motor_Y_Flag_Update(1);
00193     /* Enable the motor control for the Z motor. */
00194     Motor_Control_Motor_Z_Flag_Update(1);
00195     /* Enable Timer 0 to update the motor control. */
00196     Timer0_Enable();
00197     /* Enable Timer 0 to update the swarm algorithm. */
00198     Timer1_Enable();
00199 }
00200
00201 void State_Machine_Trasition_5(void)
00202 {
00203     #if Print_State == 1
00204         USART_Send_String("State transition 5\n\r");
00205     #endif
00206     /* Disable the turning motor. */
00207     DRV8830_Set_Voltage_Motor_X(0.0);
00208     /* Set the forward motor to maximum voltage. */
00209     DRV8830_Set_Voltage_Motor_Y(Motor_Max_Voltage);
00210     /* Disable the motor control for the X motor. */
00211     Motor_Control_Motor_X_Flag_Update(0);
00212     /* Disable the motor control for the Y motor. */
00213     Motor_Control_Motor_Y_Flag_Update(0);
00214     /* Enable the motor control for the Z motor. */
00215     Motor_Control_Motor_Z_Flag_Update(1);
00216     /* Enable Timer 0 to update the motor control. */
00217     Timer0_Enable();
00218     /* Disable Timer 1. */
00219     Timer1_Disable();
00220 }
00221
00222 void State_Machine_Trasition_6(void)
00223 {
00224     #if Print_State == 1
00225         USART_Send_String("State transition 6\n\r");
00226     #endif
00227     /* Disable the turning motor. */
00228     DRV8830_Set_Voltage_Motor_X(0.0);
00229     /* Disable the forward motor. */
00230     DRV8830_Set_Voltage_Motor_Y(0.0);
00231     /* Set the motor control to surface. */
00232     Motor_Control_Update_Depth_Z(0);
00233     /* Disable the motor control for the X motor. */
00234     Motor_Control_Motor_X_Flag_Update(0);
00235     /* Disable the motor control for the Y motor. */
00236     Motor_Control_Motor_Y_Flag_Update(0);
00237     /* Enable the motor control for the Z motor. */
00238     Motor_Control_Motor_Z_Flag_Update(1);
00239     /* Enable Timer 0 to update the motor control. */
00240     Timer0_Enable();
00241     /* Disable Timer 1. */
00242     Timer1_Disable();
00243 }
00244
00245 void State_Machine_Update(void)
00246 {
00247     static int Current_State = 0;
00248     static int Previous_State = 0;
00249     if(Update_State_Flag)
00250     {
00251         switch(Current_State)
00252         {
00253             case 0:
00254                 State_Machine_Initialization();
00255                 #if State_Machine_Test_Mode > 0
00256                 Current_State = State_Machine_Test_Mode;

```

```

00257         #else
00258             Current_State = 1;
00259         #endif
00260         break;
00261     case 1:
00262         State_Machine_State_1();
00263         #if Print_State == 1
00264             char Buffer_State_1[30];
00265             char Buffer_Depth[FLOATING_POINT_BUFFER_SIZE] = {0x00};
00266             sprintf(Buffer_State_1, "%s ft\n\r", USART_ftoa(
MPX5010GP_Depth(), Buffer_Depth));
00267             USART_Send_String(Buffer_State_1);
00268         #endif
00269         if(MPX5010GP_Depth() > 0.5)
00270         {
00271             Current_State = 2;
00272         }
00273         break;
00274     case 2:
00275         State_Machine_State_2();
00276         uint8_t IsStabilized = LSM303_IsStablized();
00277         #if Print_State == 1
00278             if(IsStabilized)
00279             {
00280                 USART_Send_String("Is stabilized");
00281             }
00282             else
00283             {
00284                 USART_Send_String("Is not stabilized");
00285             }
00286         #endif
00287         if(IsStabilized)
00288         {
00289             Current_State = 3;
00290         }
00291         break;
00292     case 3:
00293         State_Machine_State_3();
00294         #if Print_State == 1
00295             if(LSM303_RotationFinished())
00296             {
00297                 USART_Send_String("Rotation is finished\n");
00298             }
00299             else
00300             {
00301                 USART_Send_String("Rotation is not finished\n");
00302             }
00303         #endif
00304         if(LSM303_RotationFinished())
00305         {
00306             Current_State = 4;
00307         }
00308         break;
00309     case 4:
00310         State_Machine_State_4();
00311         uint8_t Motor_Status = DRV8830_Check_Motors();
00312         if(Motor_Status&0x02)
00313         {
00314             Current_State = 6;
00315         }
00316         else if(Battery_Check_Status() || (Motor_Status&0x05))
00317         {
00318             Current_State = 6;
00319         }
00320         break;
00321     case 5:
00322         State_Machine_State_5();
00323         if(CLS15_Retrieve_Photodiode_ADC_Front() <
Radius_Red)
00324         {
00325             Current_State = 6;
00326         }
00327         break;
00328     case 6:
00329         State_Machine_State_6();
00330         break;
00331 }
00332 #if State_Machine_Test_Mode == 0
00333     if(Current_State != Previous_State)
00334     {
00335         switch(Current_State)
00336         {
00337             case 2:
00338                 State_Machine_Trasition_2();
00339                 break;
00340             case 3:
00341                 State_Machine_Trasition_3();

```

```

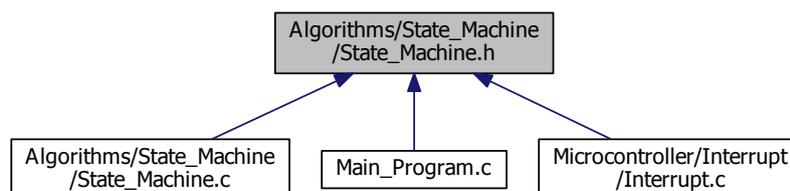
00342         break;
00343         case 4:
00344             State_Machine_Trasition_4();
00345         break;
00346         case 5:
00347             State_Machine_Trasition_5();
00348         break;
00349         case 6:
00350             State_Machine_Trasition_6();
00351         break;
00352     }
00353 }
00354 #else
00355     if(Current_State != Previous_State)
00356     {
00357         switch(Current_State)
00358         {
00359             case 2:
00360                 State_Machine_Trasition_2();
00361             break;
00362             case 3:
00363                 State_Machine_Trasition_3();
00364             break;
00365             case 4:
00366                 State_Machine_Trasition_4();
00367             break;
00368             case 5:
00369                 State_Machine_Trasition_5();
00370             break;
00371             case 6:
00372                 State_Machine_Trasition_6();
00373             break;
00374         }
00375     }
00376     else
00377     {
00378         Current_State = Previous_State;
00379     }
00380 #endif
00381 Previous_State = Current_State;
00382 Update_State_Flag = 0;
00383 #if Print_State == 1
00384 char Buffer_State_4[100];
00385 char Buffer_State_4_2[FLOATING_POINT_BUFFER_SIZE];
00386 sprintf(Buffer_State_4, "Battery Status: %d, %s V\n\r",
Battery_Check_Status(), USART_ftoa(
ADC_Convert_To_Voltage(CLS15_Retrieve_Phodiode_ADC_Front
()),Buffer_State_4_2));
00387     USART_Send_String(Buffer_State_4);
00388 #endif
00389 }
00390 }
00391
00392 void State_Machine_Enable_Flag(void)
00393 {
00394     Update_State_Flag = 1;
00395 }

```

3.11 Algorithms/State_Machine/State_Machine.h File Reference

This file is the header file for the state machine code.

This graph shows which files directly or indirectly include this file:



Functions

- void [State_Machine_Enable_Flag](#) (void)
- void [State_Machine_Update](#) (void)

3.11.1 Detailed Description

This file is the header file for the state machine code.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 4/29/2015

Created: 2/12/2015 4:00:28 PM

Definition in file [State_Machine.h](#).

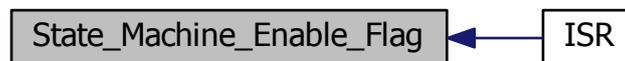
3.11.2 Function Documentation

3.11.2.1 void [State_Machine_Enable_Flag](#) (void)

Definition at line [392](#) of file [State_Machine.c](#).

Referenced by [ISR\(\)](#).

Here is the caller graph for this function:



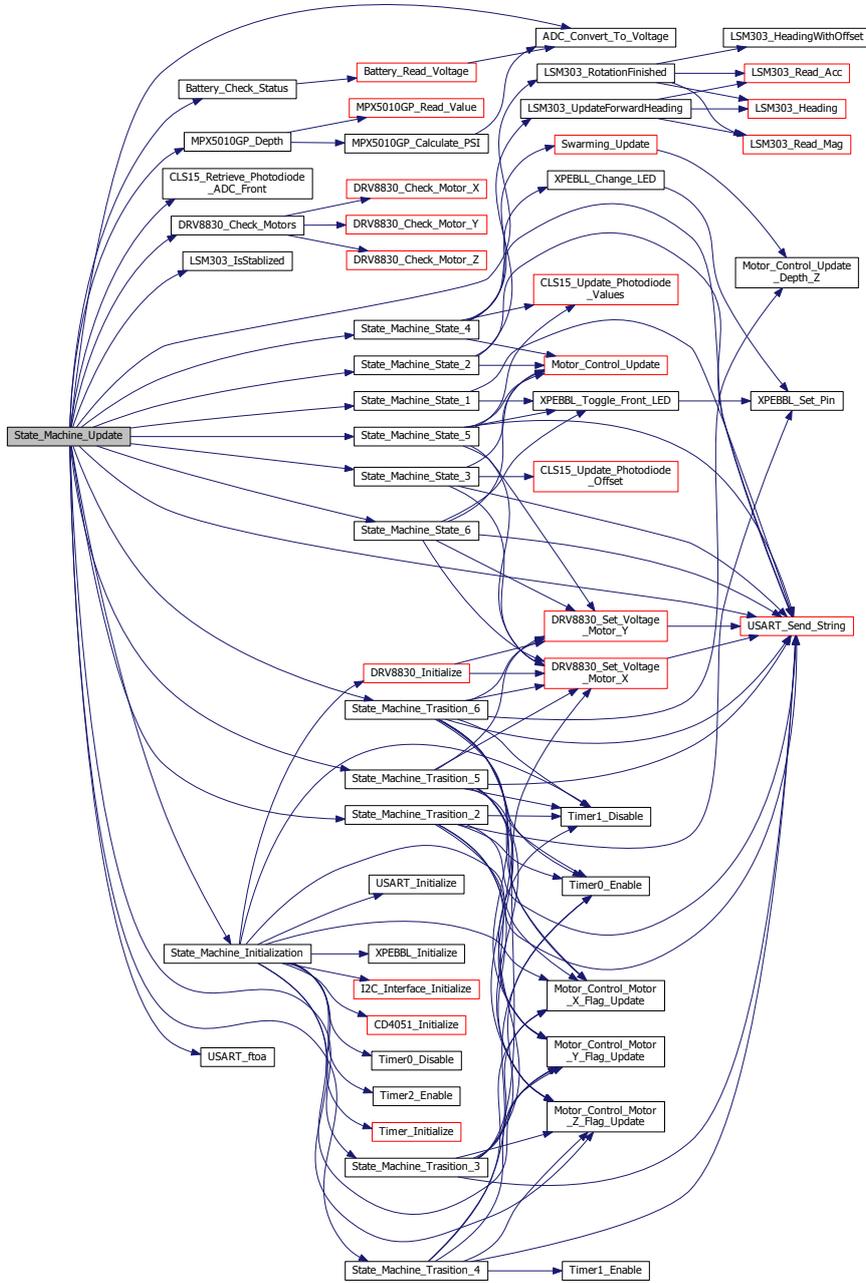
3.11.2.2 void [State_Machine_Update](#) (void)

Definition at line [245](#) of file [State_Machine.c](#).

References [ADC_Convert_To_Voltage\(\)](#), [Battery_Check_Status\(\)](#), [CLS15_Retrieve_Phodiode_ADC_Front\(\)](#), [DRV8830_Check_Motors\(\)](#), [FLOATING_POINT_BUFFER_SIZE](#), [LSM303_IsStablized\(\)](#), [LSM303_RotationFinished\(\)](#), [MPX5010GP_Depth\(\)](#), [Radius_Red](#), [State_Machine_Initialization\(\)](#), [State_Machine_State_1\(\)](#), [State_Machine_State_2\(\)](#), [State_Machine_State_3\(\)](#), [State_Machine_State_4\(\)](#), [State_Machine_State_5\(\)](#), [State_Machine_State_6\(\)](#), [State_Machine_Test_Mode](#), [State_Machine_Trasition_2\(\)](#), [State_Machine_Trasition_3\(\)](#), [State_Machine_Trasition_4\(\)](#), [State_Machine_Trasition_5\(\)](#), [State_Machine_Trasition_6\(\)](#), [USART_ftoa\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Main_Program\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.12 State_Machine.h

```

00001 /**
00002  * @file State_Machine.h
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 4/29/2015\n Created: 2/12/2015 4:00:28 PM
00006  * @brief This file is the header file for the state machine code.
00007  */
00008
00009 #ifndef STATE_MACHINE_H_
00010 #define STATE_MACHINE_H_
00011
00012 void State_Machine_Enable_Flag(void);
00013 void State_Machine_Update(void);
00014
00015 #endif /* STATE_MACHINE_H_ */

```

3.13 Main.c File Reference

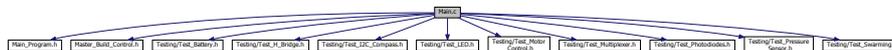
This file is code main for testing and the main program.

```

#include "Main_Program.h"
#include "Master_Build_Control.h"
#include "Testing/Test_Battery.h"
#include "Testing/Test_H_Bridge.h"
#include "Testing/Test_I2C_Compass.h"
#include "Testing/Test_LED.h"
#include "Testing/Test_Motor_Control.h"
#include "Testing/Test_Multiplexer.h"
#include "Testing/Test_Photodiodes.h"
#include "Testing/Test_Pressure_Sensor.h"
#include "Testing/Test_Swarming.h"

```

Include dependency graph for Main.c:



Functions

- int [main](#) (void)

This the main for the entire project use `Main_Mode` to select the portion of the code to test.

3.13.1 Detailed Description

This file is code main for testing and the main program.

Author

Ryan Lipski, Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 4/11/2015

Created: 11/11/2014 11:57:32 AM

Definition in file [Main.c](#).

3.13.2 Function Documentation

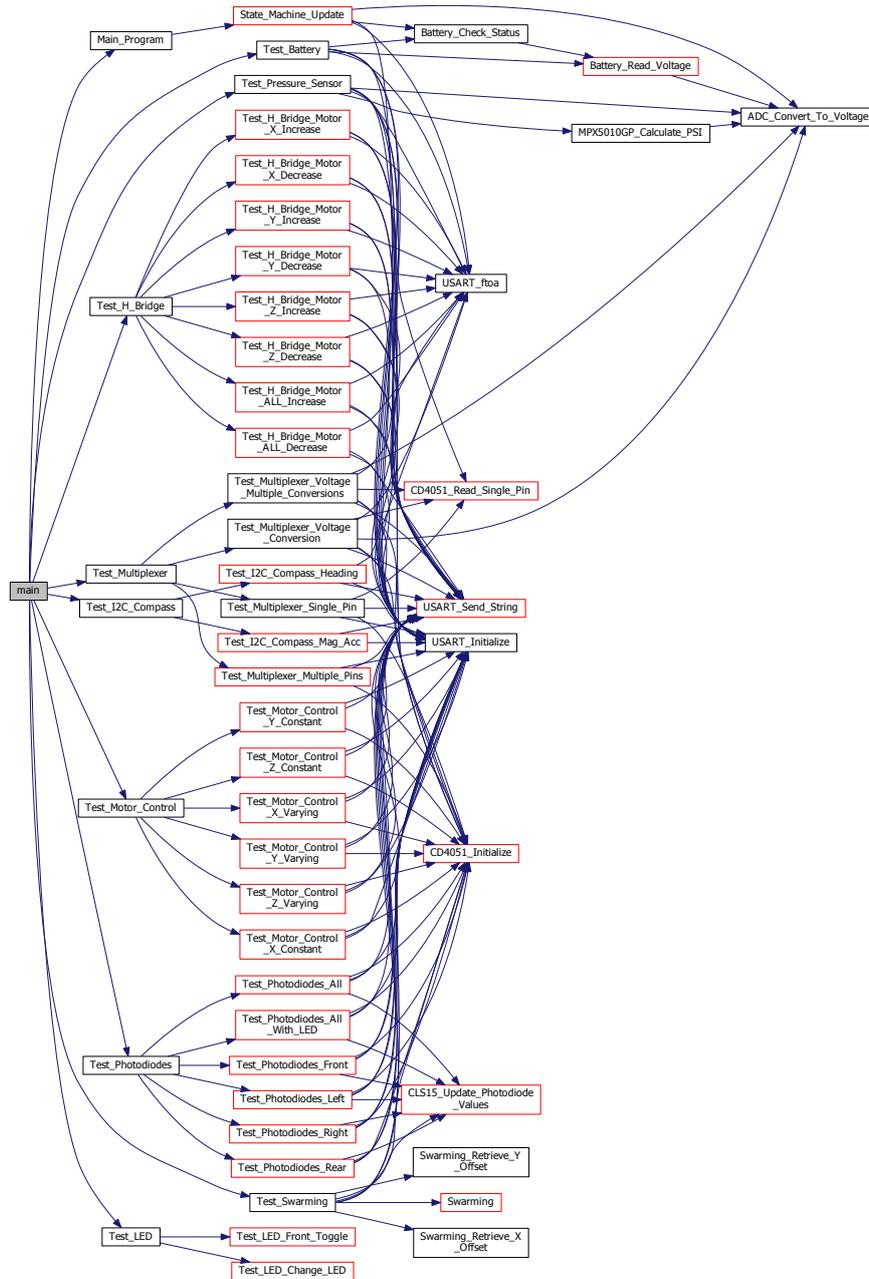
3.13.2.1 `int main (void)`

This the main for the entire project use `Main_Mode` to select the portion of the code to test.

Definition at line 24 of file [Main.c](#).

References [Main_Program\(\)](#), [Test_Battery\(\)](#), [Test_H_Bridge\(\)](#), [Test_I2C_Compass\(\)](#), [Test_LED\(\)](#), [Test_Motor_↔Control\(\)](#), [Test_Multiplexer\(\)](#), [Test_Photodiodes\(\)](#), [Test_Pressure_Sensor\(\)](#), and [Test_Swarming\(\)](#).

Here is the call graph for this function:



3.14 Main.c

```

00001 /**
00002  * @file Main.c
00003  * @author Ryan Lipski, Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 4/11/2015\n Created: 11/11/2014 11:57:32 AM
00006  * @brief This file is code main for testing and the main program.
00007  */
00008
00009 #include "Main_Program.h"
00010 #include "Master_Build_Control.h"
00011 #include "Testing/Test_Battery.h"
00012 #include "Testing/Test_H_Bridge.h"
00013 #include "Testing/Test_I2C_Compass.h"

```

```

00014 #include "Testing/Test_LED.h"
00015 #include "Testing/Test_Motor_Control.h"
00016 #include "Testing/Test_Multiplexer.h"
00017 #include "Testing/Test_Photodiodes.h"
00018 #include "Testing/Test_Pressure_Sensor.h"
00019 #include "Testing/Test_Swarming.h"
00020
00021 /**
00022  * @brief This the main for the entire project use Main_Mode to select the portion of the code to test.
00023  */
00024 int main(void)
00025 {
00026     /* Main for the entire project. */
00027     #if Main_Mode == 1
00028         Main_Program();
00029     /* Test the swarming code. */
00030     #elif Main_Mode == 2
00031         Test_Swarming();
00032     /* Test the LSM303 (I2C Compass/Accelerometer). */
00033     #elif Main_Mode == 3
00034         Test_I2C_Compass();
00035     /* Test the DRV8830 (I2C H-Bridge). */
00036     #elif Main_Mode == 4
00037         Test_H_Bridge();
00038     /* Test the LED toggling. */
00039     #elif Main_Mode == 5
00040         Test_LED();
00041     /* Test the Multiplexer and the ADC. */
00042     #elif Main_Mode == 6
00043         Test_Multiplexer();
00044     /* Test the photodiode circuits. */
00045     #elif Main_Mode == 7
00046         Test_Photodiodes();
00047     /* Test the pressure sensor. */
00048     #elif Main_Mode == 8
00049         Test_Pressure_Sensor();
00050     /* Test the battery reading code. */
00051     #elif Main_Mode == 9
00052         Test_Battery();
00053     /* Test the motor control. */
00054     #elif Main_Mode == 10
00055         Test_Motor_Control();
00056     /* Test the swarming code. */
00057     #elif Main_Mode == 11
00058         Test_Swarming();
00059     #endif
00060     /* Return of 0 for main. */
00061     return(0);
00062 }

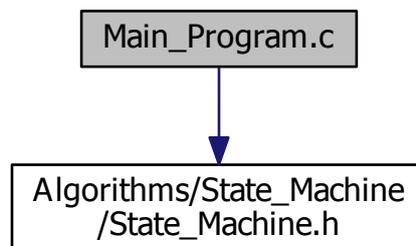
```

3.15 Main_Program.c File Reference

This file is code for the main program.

```
#include "Algorithms/State_Machine/State_Machine.h"
```

Include dependency graph for Main_Program.c:



Functions

- void [Main_Program](#) (void)

This is the main program for the project.

3.15.1 Detailed Description

This file is code for the main program.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 3/20/2015
Created: 2/28/2015 5:50:34 PM

Definition in file [Main_Program.c](#).

3.15.2 Function Documentation

3.15.2.1 void [Main_Program](#) (void)

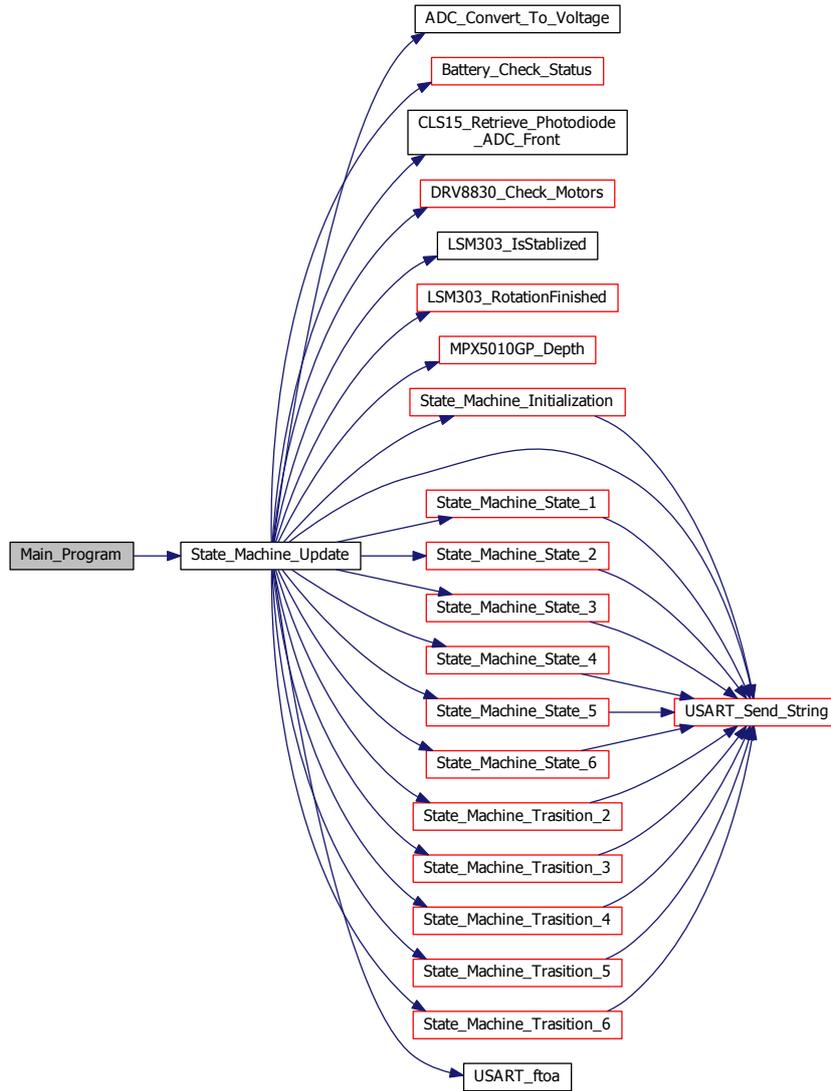
This is the main program for the project.

Definition at line 14 of file [Main_Program.c](#).

References [State_Machine_Update\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.16 Main_Program.c

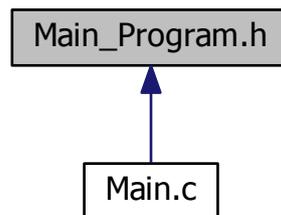
00001 /**

```
00002 * @file Main_Program.c
00003 * @author Nicholas Sikkema
00004 * @version Revision: 1.0
00005 * @date Last Updated: 3/20/2015\n Created: 2/28/2015 5:50:34 PM
00006 * @brief This file is code for the main program.
00007 */
00008
00009 #include "Algorithms/State_Machine/State_Machine.h"
00010
00011 /**
00012 * @brief This is the main program for the project.
00013 */
00014 void Main_Program(void)
00015 {
00016     while(1)
00017     {
00018         /* Check to see if the state machine needs to be updated. */
00019         State_Machine_Update();
00020     }
00021 }
```

3.17 Main_Program.h File Reference

This is the header file for the main program.

This graph shows which files directly or indirectly include this file:



Functions

- void [Main_Program](#) (void)
This is the main program for the project.

3.17.1 Detailed Description

This is the header file for the main program.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 3/20/2015
Created: 2/28/2015 5:58:00 PM

Definition in file [Main_Program.h](#).

3.17.2 Function Documentation

3.17.2.1 void Main_Program (void)

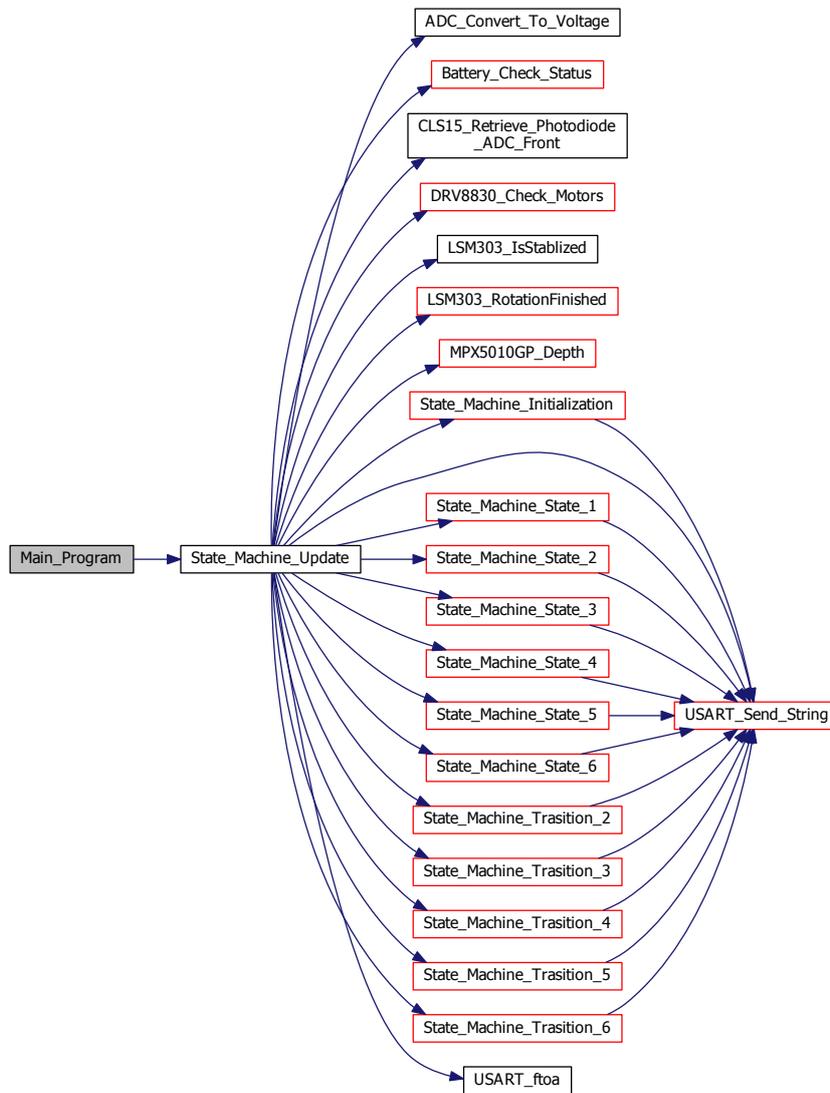
This is the main program for the project.

Definition at line 14 of file [Main_Program.c](#).

References [State_Machine_Update\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.18 Main_Program.h

```

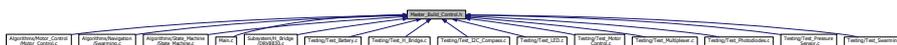
00001 /**
00002  * @file Main_Program.h
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 3/20/2015\n Created: 2/28/2015 5:58:00 PM
00006  * @brief This is the header file for the main program.
00007  */
00008
00009 #ifndef MAIN_PROGRAM_H_
00010 #define MAIN_PROGRAM_H_
00011
00012 void Main_Program(void);
00013
00014 #endif /* MAIN_PROGRAM_H_ */
  
```

3.19 mainpage.dox File Reference

3.20 Master_Build_Control.h File Reference

This is the header file to control which program builds.

This graph shows which files directly or indirectly include this file:



Macros

- `#define Main_Mode 1`
Different modes for the main function
Mode 1: Main for the entire project.
Mode 2: Test the swarming code.
Mode 3: Test the LSM303 (I2C Compass/Accelerometer).
Mode 4: Test the DRV8830 (I2C H-Bridge).
Mode 5: Test the LED toggling.
Mode 6: Test the Multiplexer and the ADC.
Mode 7: Test the photodiode circuits.
Mode 8: Test the pressure sensor.
Mode 9: Test the battery reading code.
Mode 10: Test the motor control.
Mode 11: Test the swarming code.
- `#define Compass_Test_Mode 0`

Different modes for the compass function
Compass Test Mode 0: Test Acceleration and Magnetometer Readings
Compass Test Mode 1: Test Heading

- #define [H_Bridge_Test_Mode](#) 0

Different modes for the h-bridge test function
H-Bridge Test Mode 0: Test X-axis
H-Bridge Test Mode 1: Test Y-axis
H-Bridge Test Mode 2: Test Z-axis

- #define [H_Bridge_Test_Direction](#) 0

Different modes for the h-bridge test function
H-Bridge Test Direction 0: Test goes from zero to max then repeats
H-Bridge Test Direction 1: Test goes from max to zero then repeats

- #define [LED_Test_Mode](#) 0

Different modes for the LED function
LED Test Mode 0: Test the toggling of the front LED
LED Test Mode 1: Test changing between 3 LEDs

- #define [Multiplexer_Test_Mode](#) 0

Different modes for the multiplexer function
Multiplexer Test Mode 0: Test a single pin
Multiplexer Test Mode 1: Test all of the pins
Multiplexer Test Mode 2: Test a single voltage conversion
Multiplexer Test Mode 3: Test voltage conversion on all of the pins

- #define [Photodiode_Test_Mode](#) 0

Different modes for the photodiode function
Photodiode Test Mode 0: Test front diode
Photodiode Test Mode 1: Test left side diodes
Photodiode Test Mode 2: Test right side diodes
Photodiode Test Mode 3: Test rear diode
Photodiode Test Mode 4: Test all diodes
Photodiode Test Mode 5: Test all diodes and change the LEDs

- #define [Motor_Control_Test_Mode](#) 0

Different modes for the motor control test function.
Motor Control Test Mode 0: Varying Test Forward/Reverse
Motor Control Test Mode 1: Varying Test Left/Right
Motor Control Test Mode 2: Varying Test Up/Down
Motor Control Test Mode 0: Constant Test Forward/Reverse
Motor Control Test Mode 1: Constant Test Left/Right
Motor Control Test Mode 2: Constant Test Up/Down

- #define [Print_Region](#) 0

Prints the different regions for the swarming code. 1: On / 0: Off.

- #define [Print_Motor_Information](#) 0

Prints the information for motor control code. 1: On / 0: Off.

- #define [Print_State](#) 0

Prints the different state information for the state machine code. 1: On / 0: Off.

- #define [H_Bridge_Print](#) 0

Prints the different information for the h bridge update code. 1 for on 0 for off.

- #define [State_Machine_Test_Mode](#) 0

Selects which state to test after the initialization. Note that this has to be > 0.

3.20.1 Detailed Description

This is the header file to control which program builds.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 3/20/2015

Created: 2/26/2015 4:23:20 PM

Definition in file [Master_Build_Control.h](#).

3.20.2 Macro Definition Documentation

3.20.2.1 #define Compass_Test_Mode 0

Different modes for the compass function

Compass Test Mode 0: Test Acceleration and Magnetometer Readings

Compass Test Mode 1: Test Heading

.

Definition at line 32 of file [Master_Build_Control.h](#).

3.20.2.2 #define H_Bridge_Print 0

Prints the different information for the h bridge update code. 1 for on 0 for off.

Definition at line 95 of file [Master_Build_Control.h](#).

3.20.2.3 #define H_Bridge_Test_Direction 0

Different modes for the h-bridge test function

H-Bridge Test Direction 0: Test goes from zero to max then repeats

H-Bridge Test Direction 1: Test goes from max to zero then repeats

.

Definition at line 45 of file [Master_Build_Control.h](#).

3.20.2.4 #define H_Bridge_Test_Mode 0

Different modes for the h-bridge test function

H-Bridge Test Mode 0: Test X-axis

H-Bridge Test Mode 1: Test Y-axis

H-Bridge Test Mode 2: Test Z-axis

.

Definition at line 39 of file [Master_Build_Control.h](#).

3.20.2.5 #define LED_Test_Mode 0

Different modes for the LED function

LED Test Mode 0: Test the toggling of the front LED

LED Test Mode 1: Test changing between 3 LEDs

.

Definition at line 51 of file [Master_Build_Control.h](#).

3.20.2.6 #define Main_Mode 1

Different modes for the main function

Mode 1: Main for the entire project.

Mode 2: Test the swarming code.

Mode 3: Test the LSM303 (I2C Compass/Accelerometer).

Mode 4: Test the DRV8830 (I2C H-Bridge).

Mode 5: Test the LED toggling.

Mode 6: Test the Multiplexer and the ADC.

Mode 7: Test the photodiode circuits.

Mode 8: Test the pressure sensor.

Mode 9: Test the battery reading code.

Mode 10: Test the motor control.

Mode 11: Test the swarming code.

.

Definition at line 26 of file [Master_Build_Control.h](#).

3.20.2.7 #define Motor_Control_Test_Mode 0

Different modes for the motor control test function.

Motor Control Test Mode 0: Varying Test Forward/Reverse

Motor Control Test Mode 1: Varying Test Left/Right

Motor Control Test Mode 2: Varying Test Up/Down

Motor Control Test Mode 0: Constant Test Forward/Reverse

Motor Control Test Mode 1: Constant Test Left/Right

Motor Control Test Mode 2: Constant Test Up/Down

.

Definition at line 79 of file [Master_Build_Control.h](#).

3.20.2.8 #define Multiplexer_Test_Mode 0

Different modes for the multiplexer function

Multiplexer Test Mode 0: Test a single pin

Multiplexer Test Mode 1: Test all of the pins

Multiplexer Test Mode 2: Test a single voltage conversion

Multiplexer Test Mode 3: Test voltage conversion on all of the pins

.

Definition at line 59 of file [Master_Build_Control.h](#).

3.20.2.9 #define Photodiode_Test_Mode 0

Different modes for the photodiode function

Photodiode Test Mode 0: Test front diode

Photodiode Test Mode 1: Test left side diodes

Photodiode Test Mode 2: Test right side diodes

Photodiode Test Mode 3: Test rear diode
 Photodiode Test Mode 4: Test all diodes
 Photodiode Test Mode 5: Test all diodes and change the LEDs

Definition at line 69 of file [Master_Build_Control.h](#).

3.20.2.10 #define Print_Motor_Information 0

Prints the information for motor control code. 1: On / 0: Off.

Definition at line 87 of file [Master_Build_Control.h](#).

3.20.2.11 #define Print_Region 0

Prints the different regions for the swarming code. 1: On / 0: Off.

Definition at line 83 of file [Master_Build_Control.h](#).

3.20.2.12 #define Print_State 0

Prints the different state information for the state machine code. 1: On / 0: Off.

Definition at line 91 of file [Master_Build_Control.h](#).

3.20.2.13 #define State_Machine_Test_Mode 0

Selects which state to test after the initialization. Note that this has to be > 0.

Definition at line 99 of file [Master_Build_Control.h](#).

Referenced by [State_Machine_Update\(\)](#).

3.21 Master_Build_Control.h

```
00001 /**
00002  * @file Master_Build_Control.h
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 3/20/2015\n Created: 2/26/2015 4:23:20 PM
00006  * @brief This is the header file to control which program builds.
00007  */
00008
00009 #ifndef MASTER_BUILD_CONTROL_H_
00010 #define MASTER_BUILD_CONTROL_H_
00011
00012 /**
00013  * @brief Different modes for the main function\n
00014  * Mode 1: Main for the entire project.\n
00015  * Mode 2: Test the swarming code.\n
00016  * Mode 3: Test the LSM303 (I2C Compass/Accelerometer).\n
00017  * Mode 4: Test the DRV8830 (I2C H-Bridge).\n
00018  * Mode 5: Test the LED toggling.\n
00019  * Mode 6: Test the Multiplexer and the ADC.\n
00020  * Mode 7: Test the photodiode circuits.\n
00021  * Mode 8: Test the pressure sensor.\n
00022  * Mode 9: Test the battery reading code.\n
00023  * Mode 10: Test the motor control.\n
00024  * Mode 11: Test the swarming code.\n
00025  */
00026 #define Main_Mode 1
00027 /**
00028  * @brief Different modes for the compass function\n
00029  * Compass Test Mode 0: Test Acceleration and Magnetometer Readings\n
00030  * Compass Test Mode 1: Test Heading\n
00031  */
00032 #define Compass_Test_Mode 0
00033 /**
```

```

00034 * @brief Different modes for the h-bridge test function\n
00035 * H-Bridge Test Mode 0: Test X-axis\n
00036 * H-Bridge Test Mode 1: Test Y-axis\n
00037 * H-Bridge Test Mode 2: Test Z-axis\n
00038 */
00039 #define H_Bridge_Test_Mode 0
00040 /**
00041 * @brief Different modes for the h-bridge test function\n
00042 * H-Bridge Test Direction 0: Test goes from zero to max then repeats\n
00043 * H-Bridge Test Direction 1: Test goes from max to zero then repeats\n
00044 */
00045 #define H_Bridge_Test_Direction 0
00046 /**
00047 * @brief Different modes for the LED function\n
00048 * LED Test Mode 0: Test the toggling of the front LED\n
00049 * LED Test Mode 1: Test changing between 3 LEDs\n
00050 */
00051 #define LED_Test_Mode 0
00052 /**
00053 * @brief Different modes for the multiplexer function\n
00054 * Multiplexer Test Mode 0: Test a single pin\n
00055 * Multiplexer Test Mode 1: Test all of the pins\n
00056 * Multiplexer Test Mode 2: Test a single voltage conversion\n
00057 * Multiplexer Test Mode 3: Test voltage conversion on all of the pins\n
00058 */
00059 #define Multiplexer_Test_Mode 0
00060 /**
00061 * @brief Different modes for the photodiode function\n
00062 * Photodiode Test Mode 0: Test front diode\n
00063 * Photodiode Test Mode 1: Test left side diodes\n
00064 * Photodiode Test Mode 2: Test right side diodes\n
00065 * Photodiode Test Mode 3: Test rear diode\n
00066 * Photodiode Test Mode 4: Test all diodes\n
00067 * Photodiode Test Mode 5: Test all diodes and change the LEDs\n
00068 */
00069 #define Photodiode_Test_Mode 0
00070 /**
00071 * @brief Different modes for the motor control test function.\n
00072 * Motor Control Test Mode 0: Varying Test Forward/Reverse\n
00073 * Motor Control Test Mode 1: Varying Test Left/Right\n
00074 * Motor Control Test Mode 2: Varying Test Up/Down\n
00075 * Motor Control Test Mode 0: Constant Test Forward/Reverse\n
00076 * Motor Control Test Mode 1: Constant Test Left/Right\n
00077 * Motor Control Test Mode 2: Constant Test Up/Down\n
00078 */
00079 #define Motor_Control_Test_Mode 0
00080 /**
00081 * @brief Prints the different regions for the swarming code. 1: On / 0: Off
00082 */
00083 #define Print_Region 0
00084 /**
00085 * @brief Prints the information for motor control code. 1: On / 0: Off
00086 */
00087 #define Print_Motor_Information 0
00088 /**
00089 * @brief Prints the different state information for the state machine code. 1: On / 0: Off
00090 */
00091 #define Print_State 0
00092 /**
00093 * @brief Prints the different information for the h bridge update code. 1 for on 0 for off.
00094 */
00095 #define H_Bridge_Print 0
00096 /**
00097 * @brief Selects which state to test after the initialization. Note that this has to be > 0.
00098 */
00099 #define State_Machine_Test_Mode 0
00100 #endif /* MASTER_BUILD_CONTROL_H_ */

```

3.22 Microcontroller/ADC/ADC.c File Reference

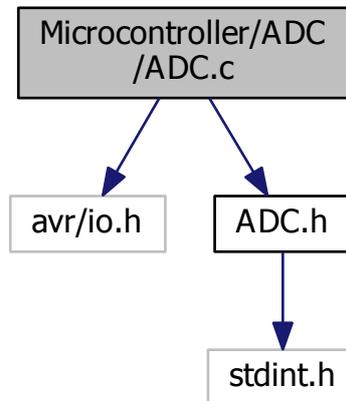
This file is code to control the ADC on the ATmega328P.

```

#include <avr/io.h>
#include "ADC.h"

```

Include dependency graph for ADC.c:



Macros

- `#define VREF 5.0`
The reference voltage for the ATmega328P.

Functions

- void `ADC_Setup_Pin` (unsigned char `ADC_Pin`)
This function sets up the ADC Pin that the function `ADC_Read_Pin` will read.
- int16_t `ADC_Read_Pin` (void)
Reads the selected pin from the function `ADC_Setup_Pin`.
- float `ADC_Convert_To_Voltage` (int16_t `ADC_Value`)
Converts the ADC value to float.

3.22.1 Detailed Description

This file is code to control the ADC on the ATmega328P.

Author

Nicholas Sikkema

Version

Revision: 1.5

Date

Last Updated: 2/19/2015
Created: 11/12/2014 7:36:35 PM

Definition in file [ADC.c](#).

3.22.2 Macro Definition Documentation

3.22.2.1 #define VREF 5.0

The reference voltage for the ATmega328P.

Definition at line 10 of file [ADC.c](#).

Referenced by [ADC_Convert_To_Voltage\(\)](#).

3.22.3 Function Documentation

3.22.3.1 float ADC_Convert_To_Voltage (int16_t ADC_Value)

Converts the ADC value to float.

Parameters

<i>ADC_Value</i>	The current ADC value from 0 to 1023.
------------------	---------------------------------------

Returns

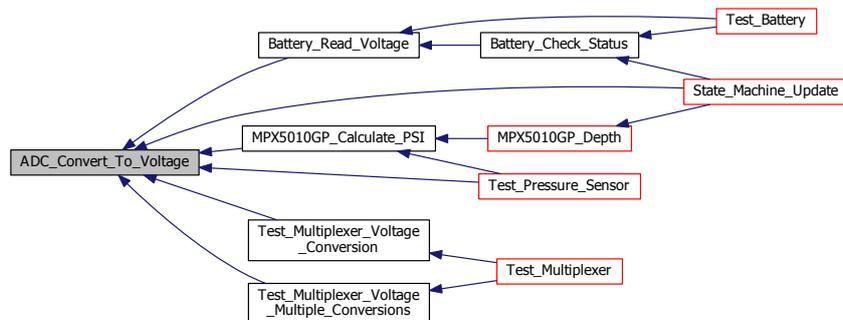
The current ADC value in voltage from 0 to $V_{ref} * 1023 / 1024$.

Definition at line 74 of file [ADC.c](#).

References [VREF](#).

Referenced by [Battery_Read_Voltage\(\)](#), [MPX5010GP_Calculate_PSI\(\)](#), [State_Machine_Update\(\)](#), [Test_Multiplexer_Voltage_Conversion\(\)](#), [Test_Multiplexer_Voltage_Multiple_Conversions\(\)](#), and [Test_Pressure_Sensor\(\)](#).

Here is the caller graph for this function:



3.22.3.2 int16_t ADC_Read_Pin (void)

Reads the selected pin from the function [ADC_Setup_Pin](#).

Returns

The current ADC value from 0 to 1023.

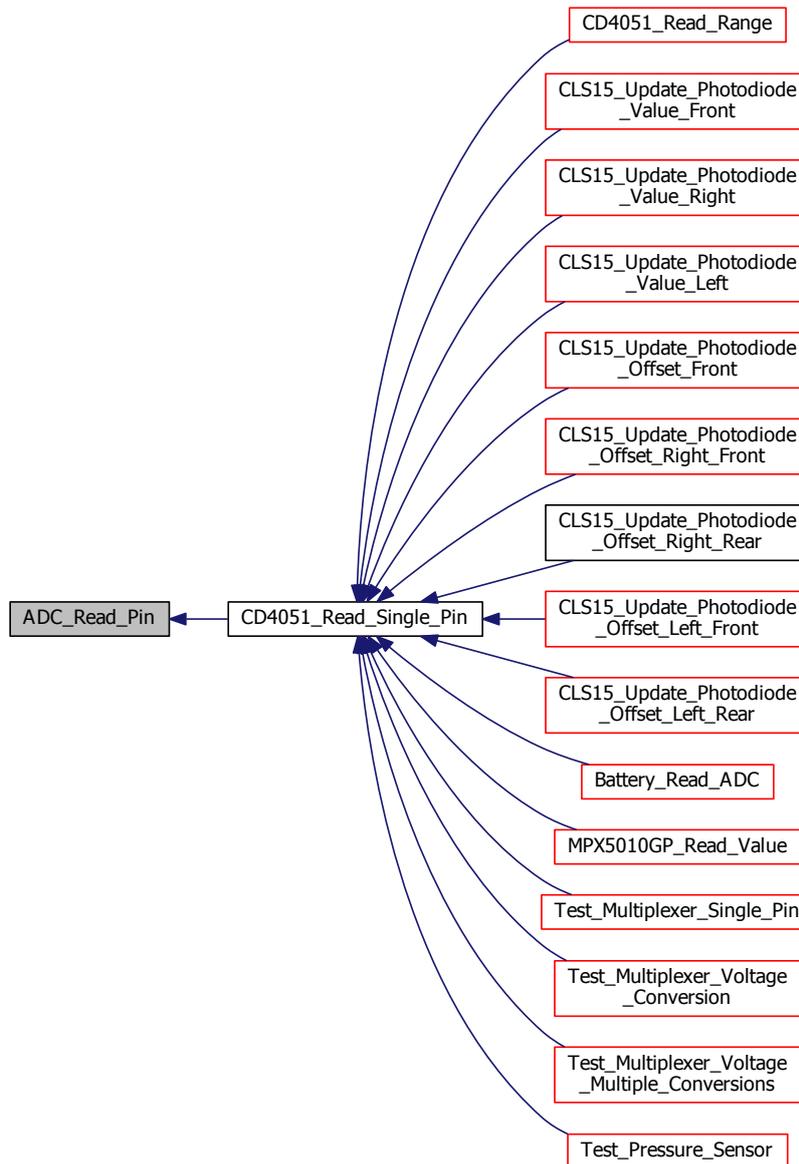
Warning

ADC_Setup_Pin needs to be ran before this function.

Definition at line 55 of file ADC.c.

Referenced by CD4051_Read_Single_Pin().

Here is the caller graph for this function:



3.22.3.3 void ADC_Setup_Pin (unsigned char ADC_Pin)

This function sets up the ADC Pin that the function ADC_Read_Pin will read.

Parameters

<i>ADC_Pin</i>	Pin to be set up.
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	ADC8
1001	(reserved)
1010	(reserved)
1011	(reserved)
1100	(reserved)
1101	(reserved)
1110	1.1V (V _{BG})
1111	0V (GND)

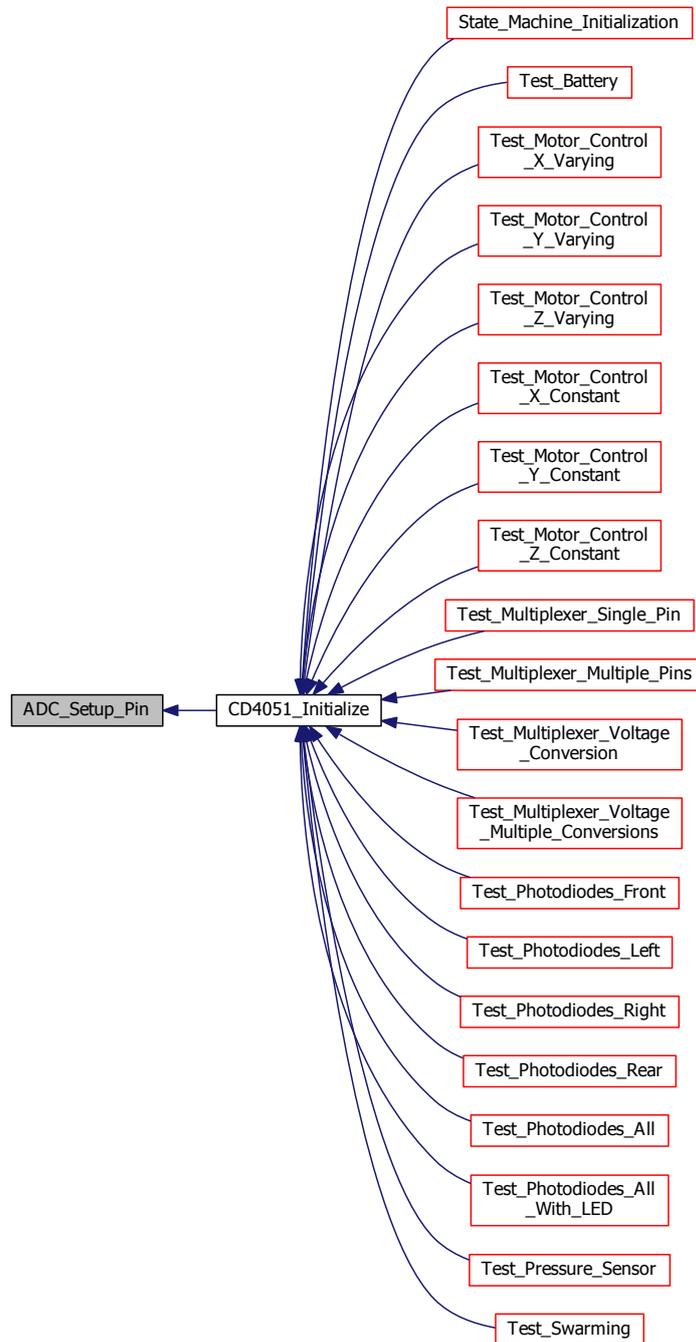
Warning

This function does not remove the upper 4 bits from the input.

Definition at line 36 of file [ADC.c](#).

Referenced by [CD4051_Initialize\(\)](#).

Here is the caller graph for this function:



3.23 ADC.c

```

00001 /**
00002  * @file ADC.c
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.5
00005  * @date Last Updated: 2/19/2015\n Created: 11/12/2014 7:36:35 PM
00006  * @brief This file is code to control the ADC on the ATmega328P.
00007  */
  
```

```

00008
00009 /** @brief The reference voltage for the ATmega328P. */
00010 #define VREF 5.0
00011
00012 #include <avr/io.h>
00013 #include "ADC.h"
00014
00015 /**
00016  * @brief This function sets up the ADC Pin that the function ADC_Read_Pin will read.
00017  * @param ADC_Pin Pin to be set up.\n
00018  * 0000 | ADC0\n
00019  * 0001 | ADC1\n
00020  * 0010 | ADC2\n
00021  * 0011 | ADC3\n
00022  * 0100 | ADC4\n
00023  * 0101 | ADC5\n
00024  * 0110 | ADC6\n
00025  * 0111 | ADC7\n
00026  * 1000 | ADC8\n
00027  * 1001 | (reserved)\n
00028  * 1010 | (reserved)\n
00029  * 1011 | (reserved)\n
00030  * 1100 | (reserved)\n
00031  * 1101 | (reserved)\n
00032  * 1110 | 1.1V (V BG)\n
00033  * 1111 | 0V (GND)\n
00034  * @warning This function does not remove the upper 4 bits from the input.
00035  */
00036 void ADC_Setup_Pin(unsigned char ADC_Pin)
00037 {
00038     /* Set up which analog input pin to use. */
00039     ADMUX = ADC_Pin;
00040     /* Use AVcc as the reference. */
00041     ADMUX |= (1<<REFS0);
00042     /* Clear for 10 bit resolution. */
00043     ADMUX &= ~(1<<ADLAR);
00044     /* Set the prescale to 128 for 8Mhz. */
00045     ADCSRA |= (1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);
00046     /* Enable the ADC. */
00047     ADCSRA |= (1<<ADEN);
00048 }
00049
00050 /**
00051  * @brief Reads the selected pin from the function ADC_Setup_Pin.
00052  * @return The current ADC value from 0 to 1023.
00053  * @warning ADC_Setup_Pin needs to be ran before this function.
00054  */
00055 int16_t ADC_Read_Pin(void)
00056 {
00057     /* Start the ADC conversion. */
00058     ADCSRA |= (1<<ADSC);
00059     /* Wait till the ADC complete flag is set. */
00060     while(ADCSRA&(1<<ADSC));
00061     /* Gather the lower byte of of the ADC value. */
00062     uint16_t ADC_Value = ADCL;
00063     /* Gather the upper byte of of the ADC value. */
00064     ADC_Value |= (ADCH<<8);
00065     /* Return the ADC value. */
00066     return(ADC_Value);
00067 }
00068
00069 /**
00070  * @brief Converts the ADC value to flat
00071  * @param ADC_Value The current ADC value from 0 to 1023.
00072  * @return The current ADC value in voltage from 0 to Vref*1023/1024.
00073  */
00074 float ADC_Convert_To_Voltage(int16_t ADC_Value)
00075 {
00076     /* Convert the ADC value to voltage */
00077     return(((float)ADC_Value*VREF)/1024.0);
00078 }

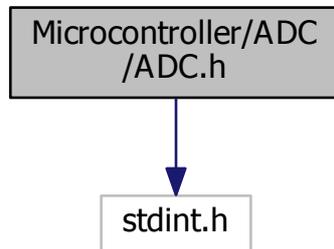
```

3.24 Microcontroller/ADC/ADC.h File Reference

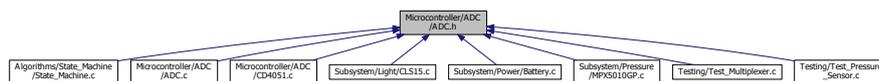
This is the header file for the code that controls the ADC on the ATmega328P.

```
#include <stdint.h>
```

Include dependency graph for ADC.h:



This graph shows which files directly or indirectly include this file:



Functions

- void [ADC_Setup_Pin](#) (unsigned char ADC_Pin)
This function sets up the ADC Pin that the function ADC_Read_Pin will read.
- int16_t [ADC_Read_Pin](#) (void)
Reads the selected pin from the function ADC_Setup_Pin.
- float [ADC_Convert_To_Voltage](#) (int16_t ADC_Value)
Converts the ADC value to flat.

3.24.1 Detailed Description

This is the header file for the code that controls the ADC on the ATmega328P.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 2/16/2015
Created: 11/12/2014 7:36:35 PM

Definition in file [ADC.h](#).

3.24.2 Function Documentation

3.24.2.1 float ADC_Convert_To_Voltage (int16_t *ADC_Value*)

Converts the ADC value to float.

Parameters

<i>ADC_Value</i>	The current ADC value from 0 to 1023.
------------------	---------------------------------------

Returns

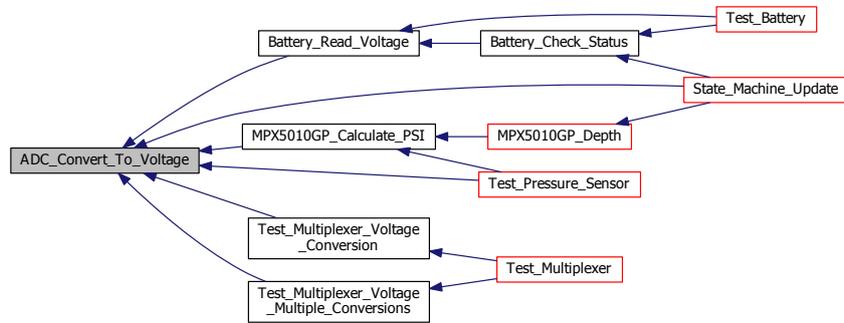
The current ADC value in voltage from 0 to Vref*1023/1024.

Definition at line 74 of file [ADC.c](#).

References [VREF](#).

Referenced by [Battery_Read_Voltage\(\)](#), [MPX5010GP_Calculate_PSI\(\)](#), [State_Machine_Update\(\)](#), [Test_Multiplexer_Voltage_Conversion\(\)](#), [Test_Multiplexer_Voltage_Multiple_Conversions\(\)](#), and [Test_Pressure_Sensor\(\)](#).

Here is the caller graph for this function:



3.24.2.2 int16_t ADC_Read_Pin (void)

Reads the selected pin from the function [ADC_Setup_Pin](#).

Returns

The current ADC value from 0 to 1023.

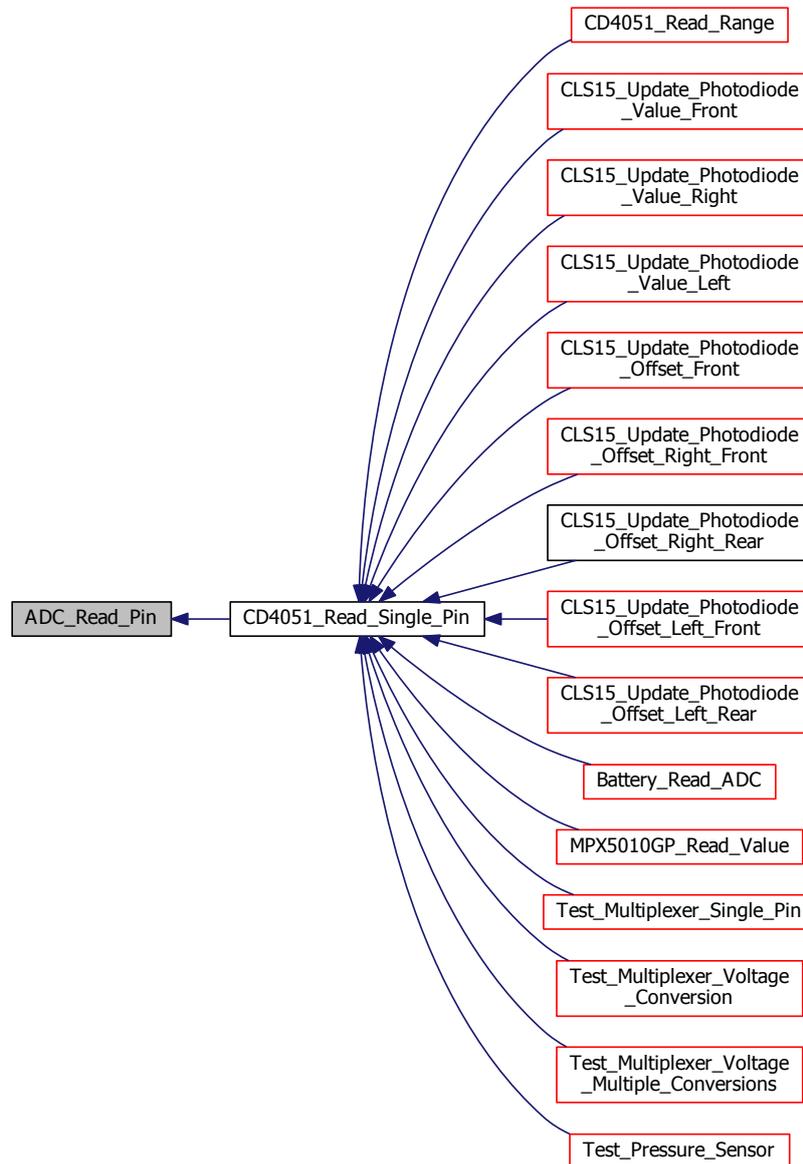
Warning

ADC_Setup_Pin needs to be ran before this function.

Definition at line 55 of file [ADC.c](#).

Referenced by [CD4051_Read_Single_Pin\(\)](#).

Here is the caller graph for this function:



3.24.2.3 void ADC_Setup_Pin (unsigned char ADC_Pin)

This function sets up the ADC Pin that the function `ADC_Read_Pin` will read.

Parameters

<i>ADC_Pin</i>	Pin to be set up.
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	ADC8
1001	(reserved)
1010	(reserved)
1011	(reserved)
1100	(reserved)
1101	(reserved)
1110	1.1V (V BG)
1111	0V (GND)

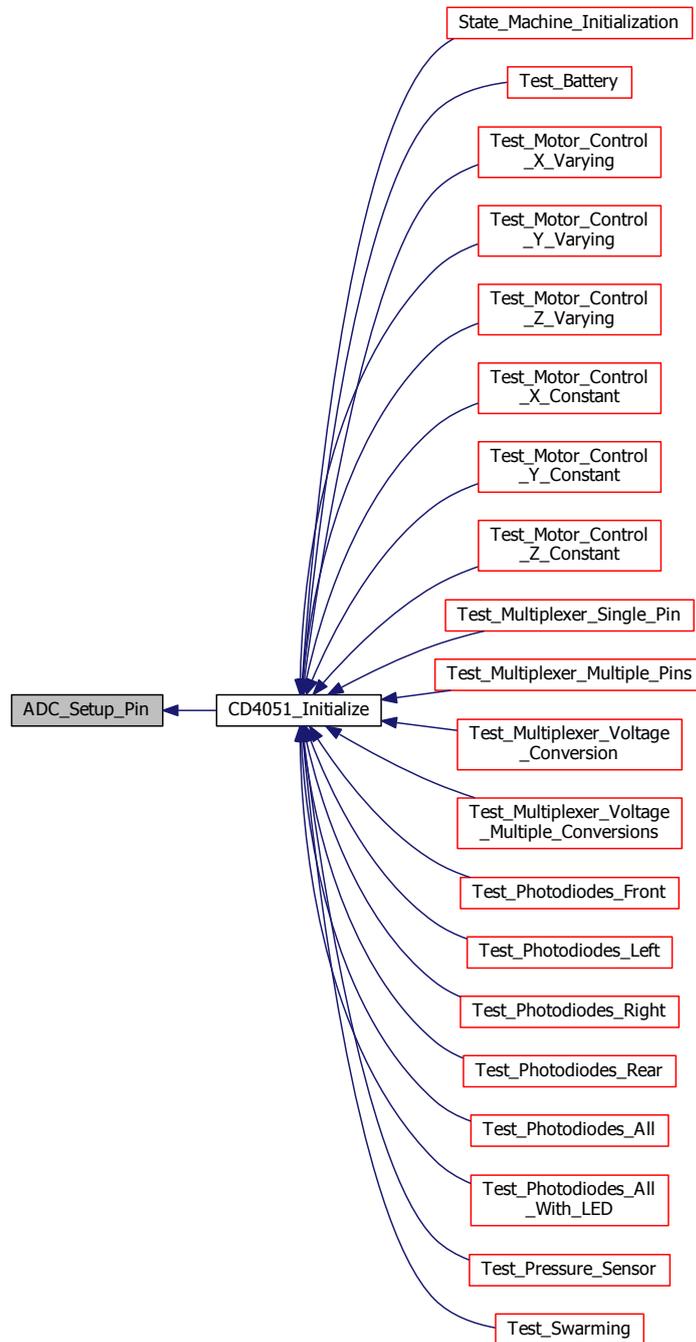
Warning

This function does not remove the upper 4 bits from the input.

Definition at line 36 of file [ADC.c](#).

Referenced by [CD4051_Initialize\(\)](#).

Here is the caller graph for this function:



3.25 ADC.h

```

00001 /**
00002  * @file ADC.h
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 2/16/2015\n Created: 11/12/2014 7:36:35 PM
00006  * @brief This is the header file for the code that controls the ADC on the ATmega328P.
00007  */
  
```

```

00008
00009 #ifndef ADC_H_
00010 #define ADC_H_
00011
00012 #include <stdint.h>
00013
00014 void ADC_Setup_Pin(unsigned char ADC_Pin);
00015 int16_t ADC_Read_Pin(void);
00016 float ADC_Convert_To_Voltage(int16_t ADC_Value);
00017
00018 #endif /* ADC_H_ */

```

3.26 Microcontroller/ADC/CD4051.c File Reference

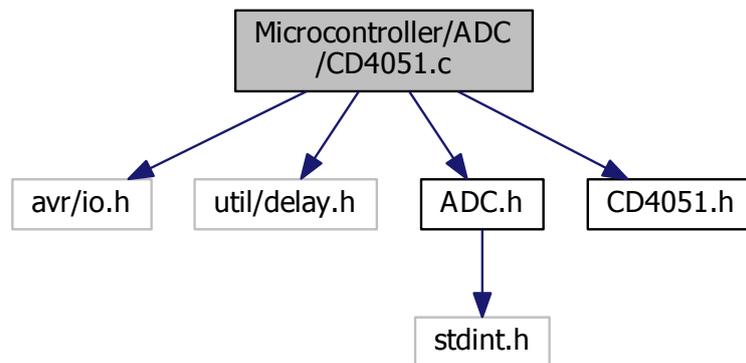
This file is code to control the CD4051 multiplexer connected to the ADC.

```

#include <avr/io.h>
#include <util/delay.h>
#include "ADC.h"
#include "CD4051.h"

```

Include dependency graph for CD4051.c:



Macros

- #define `F_CPU` 16000000UL
The current clock speed for the microcontroller.
- #define `ADC_DEFAULT_PIN` 0
The default pin for the ADC on the ATmega328P.

Functions

- void `CD4051_Initialize` (void)
This function is used to initialize the pins used for the multiplexer.
- int `CD4051_Read_Single_Pin` (unsigned char Multiplexer_Pin)
*This function is to read the selected multiplexer pin Pin 0: Voltage
Pin 1: Photodiode Front
Pin 2: Photodiode Right Front
Pin 3: Photodiode Right Rear
Pin 4: Photodiode Left Front*

Pin 5: Photodiode Left Rear
 Pin 6: Photodiode Rear
 Pin 7: Pressure Sensor

- void [CD4051_Read_Range](#) (int *Output, unsigned char Minimum_Multiplexer_Pin, unsigned char Maximum_Multiplexer_Pin)

This function is used to calculate the distance for a photodiode.

3.26.1 Detailed Description

This file is code to control the CD4051 multiplexer connected to the ADC.

Author

Nicholas Sikkema

Version

Revision: 1.5

Date

Last Updated: 1/22/2015
 Created: 10/23/2014 3:23:07 PM

Definition in file [CD4051.c](#).

3.26.2 Macro Definition Documentation

3.26.2.1 #define ADC_DEFAULT_PIN 0

The default pin for the ADC on the ATmega328P.

Definition at line 14 of file [CD4051.c](#).

Referenced by [CD4051_Initialize\(\)](#).

3.26.2.2 #define F_CPU 16000000UL

The current clock speed for the microcontroller.

Definition at line 11 of file [CD4051.c](#).

3.26.3 Function Documentation

3.26.3.1 void CD4051_Initialize (void)

This function is used to initialize the pins used for the multiplexer.

Definition at line 24 of file [CD4051.c](#).

References [ADC_DEFAULT_PIN](#), and [ADC_Setup_Pin\(\)](#).

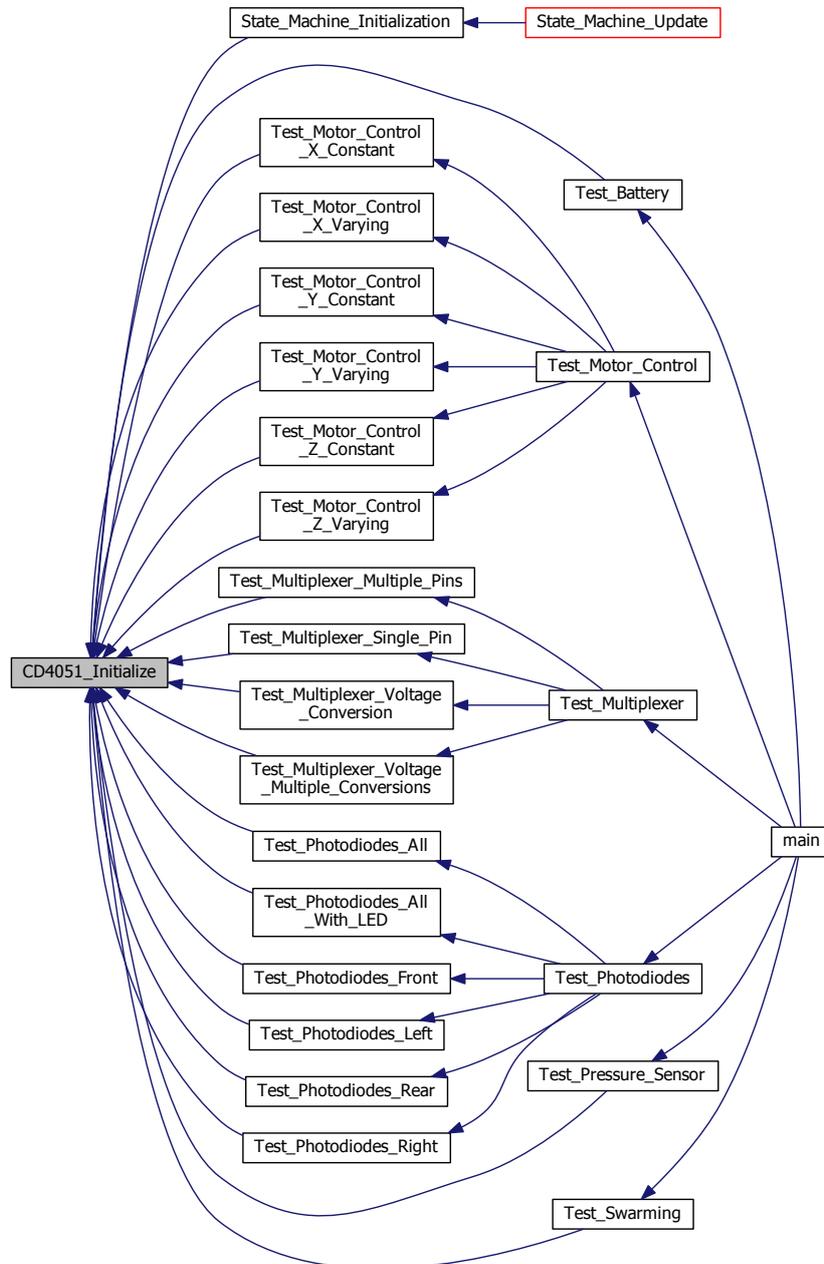
Referenced by [State_Machine_Initialization\(\)](#), [Test_Battery\(\)](#), [Test_Motor_Control_X_Constant\(\)](#), [Test_Motor_Control_X_Varying\(\)](#), [Test_Motor_Control_Y_Constant\(\)](#), [Test_Motor_Control_Y_Varying\(\)](#), [Test_Motor_Control_Z_Constant\(\)](#), [Test_Motor_Control_Z_Varying\(\)](#), [Test_Multiplexer_Multiple_Pins\(\)](#), [Test_Multiplexer_Single_Pin\(\)](#),

[Test_Multiplexer_Voltage_Conversion\(\)](#), [Test_Multiplexer_Voltage_Multiple_Conversions\(\)](#), [Test_Photodiodes_All\(\)](#), [Test_Photodiodes_All_With_LED\(\)](#), [Test_Photodiodes_Front\(\)](#), [Test_Photodiodes_Left\(\)](#), [Test_Photodiodes_Rear\(\)](#), [Test_Photodiodes_Right\(\)](#), [Test_Pressure_Sensor\(\)](#), and [Test_Swarming\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.26.3.2 void CD4051_Read_Range (int * *Output*, unsigned char *Minimum_Multiplexer_Pin*, unsigned char *Maximum_Multiplexer_Pin*)

This function is used to calculate the distance for a photodiode.

Parameters

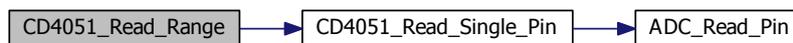
<i>Output</i>	An array holding ADC values for the selected multiplexer pins.
<i>Minimum_↔ Multiplexer_Pin</i>	Lowest pin range to be selected from the multiplexer.
<i>Maximum_↔ Multiplexer_Pin</i>	Highest pin range to be selected from the multiplexer.

Definition at line 64 of file [CD4051.c](#).

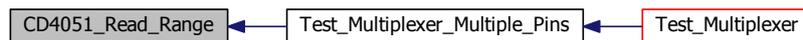
References [CD4051_Read_Single_Pin\(\)](#).

Referenced by [Test_Multiplexer_Multiple_Pins\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.26.3.3 int CD4051_Read_Single_Pin (unsigned char *Multiplexer_Pin*)

This function is to read the selected multiplexer pin Pin 0: Voltage

Pin 1: Photodiode Front

Pin 2: Photodiode Right Front

Pin 3: Photodiode Right Rear

Pin 4: Photodiode Left Front

Pin 5: Photodiode Left Rear

Pin 6: Photodiode Rear

Pin 7: Pressure Sensor

.

Parameters

<i>Multiplexer_Pin</i>	Pin to be selected from the multiplexer.
------------------------	--

Returns

ADC value for the selected multiplexer pin.

Warning

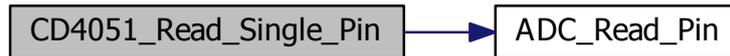
This function does not check the input for validity.

Definition at line 46 of file [CD4051.c](#).

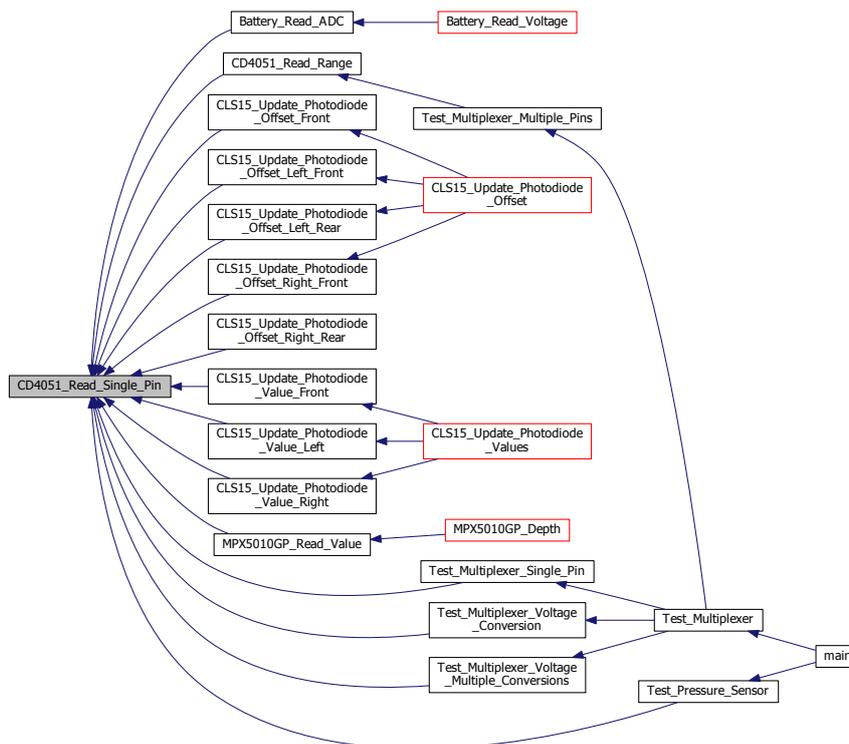
References [ADC_Read_Pin\(\)](#).

Referenced by [Battery_Read_ADC\(\)](#), [CD4051_Read_Range\(\)](#), [CLS15_Update_Photodiode_Offset_Front\(\)](#), [CLS15_Update_Photodiode_Offset_Left_Front\(\)](#), [CLS15_Update_Photodiode_Offset_Left_Rear\(\)](#), [CLS15_Update_Photodiode_Offset_Right_Front\(\)](#), [CLS15_Update_Photodiode_Offset_Right_Rear\(\)](#), [CLS15_Update_Photodiode_Value_Front\(\)](#), [CLS15_Update_Photodiode_Value_Left\(\)](#), [CLS15_Update_Photodiode_Value_Right\(\)](#), [MPX5010GP_Read_Value\(\)](#), [Test_Multiplexer_Single_Pin\(\)](#), [Test_Multiplexer_Voltage_Conversion\(\)](#), [Test_Multiplexer_Voltage_Multiple_Conversions\(\)](#), and [Test_Pressure_Sensor\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.27 CD4051.c

```

00001 /**
00002  * @file CD4051.c
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.5
00005  * @date Last Updated: 1/22/2015\n Created: 10/23/2014 3:23:07 PM
00006  * @brief This file is code to control the CD4051 multiplexer connected to the ADC.
00007  */
00008
00009 #ifndef F_CPU
  
```

```

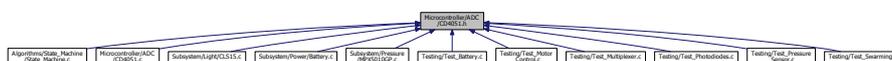
00010     /** @brief The current clock speed for the microcontroller. */
00011     #define F_CPU 16000000UL
00012 #endif
00013 /** @brief The default pin for the ADC on the ATmega328P. */
00014 #define ADC_DEFAULT_PIN 0
00015
00016 #include <avr/io.h>
00017 #include <util/delay.h>
00018 #include "ADC.h"
00019 #include "CD4051.h"
00020
00021 /**
00022  * @brief This function is used to initialize the pins used for the multiplexer
00023  */
00024 void CD4051_Initialize(void)
00025 {
00026     /* Initialize ports B1, B2, and B3 as outputs. */
00027     DDRB |= (1<<DDB2) | (1<<DDB1) | (1<<DDB0);
00028     /* Initialize the default pin for the ADC. */
00029     ADC_Setup_Pin(ADC_DEFAULT_PIN);
00030 }
00031
00032 /**
00033  * @brief This function is to read the selected multiplexer pin
00034  * Pin 0: Voltage\n
00035  * Pin 1: Photodiode Front\n
00036  * Pin 2: Photodiode Right Front\n
00037  * Pin 3: Photodiode Right Rear\n
00038  * Pin 4: Photodiode Left Front\n
00039  * Pin 5: Photodiode Left Rear\n
00040  * Pin 6: Photodiode Rear\n
00041  * Pin 7: Pressure Sensor\n
00042  * @param Multiplexer_Pin Pin to be selected from the multiplexer.
00043  * @return ADC value for the selected multiplexer pin.
00044  * @warning This function does not check the input for validity.
00045  */
00046 int CD4051_Read_Single_Pin(unsigned char Multiplexer_Pin)
00047 {
00048     /* Clear the lower three bits for PORTB. */
00049     PORTB &= 0xF8;
00050     /* Set the multiplexer pin. */
00051     PORTB |= Multiplexer_Pin;
00052     /* Have a delay to allow the switching of the multiplexer. */
00053     _delay_us(100);
00054     /* Return the ADC value for the current multiplexer pin. */
00055     return(ADC_Read_Pin());
00056 }
00057
00058 /**
00059  * @brief This function is used to calculate the distance for a photodiode.
00060  * @param Output An array holding ADC values for the selected multiplexer pins.
00061  * @param Minimum_Multiplexer_Pin Lowest pin range to be selected from the multiplexer.
00062  * @param Maximum_Multiplexer_Pin Highest pin range to be selected from the multiplexer.
00063  */
00064 void CD4051_Read_Range(int *Output, unsigned char Minimum_Multiplexer_Pin, unsigned char
Maximum_Multiplexer_Pin)
00065 {
00066     /* Initialize the index for the array. */
00067     uint8_t Index = 0;
00068     /* Loop through the range between the minimum and maximum. */
00069     for(uint8_t i = Minimum_Multiplexer_Pin; i <= Maximum_Multiplexer_Pin; i++)
00070     {
00071         /* Read the selected pin and save at the current index. */
00072         Output[Index] = CD4051_Read_Single_Pin(i);
00073         /* Increment the current index. */
00074         Index++;
00075     }
00076 }

```

3.28 Microcontroller/ADC/CD4051.h File Reference

This file is code to control the CD4051 multiplexer connected to the ADC.

This graph shows which files directly or indirectly include this file:



Macros

- #define [Battery_Mux_Port](#) 0
The port on the multiplexer that has the current reference voltage.
- #define [Photodiode_front](#) 1
Multiplexer port for the front photodiode.
- #define [Photodiode_rightfront](#) 2
Multiplexer port for the right front photodiode.
- #define [Photodiode_rightrear](#) 3
Multiplexer port for the right rear photodiode.
- #define [Photodiode_leftfront](#) 4
Multiplexer port for the left front photodiode.
- #define [Photodiode_leftrear](#) 6
Multiplexer port for the left rear photodiode.
- #define [Pressure_Sensor_Mux_Port](#) 7
The default port for the pressure sensor on the ATmega328P.

Functions

- void [CD4051_Initialize](#) (void)
This function is used to initialize the pins used for the multiplexer.
- void [CD4051_Read_Range](#) (int *Output, unsigned char Minimum_Multiplexer_Pin, unsigned char Maximum_Multiplexer_Pin)
This function is used to calculate the distance for a photodiode.
- int [CD4051_Read_Single_Pin](#) (unsigned char Multiplexer_Pin)
*This function is to read the selected multiplexer pin Pin 0: Voltage
Pin 1: Photodiode Front
Pin 2: Photodiode Right Front
Pin 3: Photodiode Right Rear
Pin 4: Photodiode Left Front
Pin 5: Photodiode Left Rear
Pin 6: Photodiode Rear
Pin 7: Pressure Sensor*

3.28.1 Detailed Description

This file is code to control the CD4051 multiplexer connected to the ADC.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 1/22/2015
Created: 10/23/2014 3:23:07 PM

Definition in file [CD4051.h](#).

3.28.2 Macro Definition Documentation

3.28.2.1 #define Battery_Mux_Port 0

The port on the multiplexer that has the current reference voltage.

Definition at line 13 of file [CD4051.h](#).

Referenced by [Battery_Read_ADC\(\)](#).

3.28.2.2 #define Photodiode_front 1

Multiplexer port for the front photodiode.

Definition at line 15 of file [CD4051.h](#).

Referenced by [CLS15_Update_Photodiode_Offset_Front\(\)](#), and [CLS15_Update_Photodiode_Value_Front\(\)](#).

3.28.2.3 #define Photodiode_leftfront 4

Multiplexer port for the left front photodiode.

Definition at line 21 of file [CD4051.h](#).

Referenced by [CLS15_Update_Photodiode_Offset_Left_Front\(\)](#), and [CLS15_Update_Photodiode_Value_Left\(\)](#).

3.28.2.4 #define Photodiode_leftrear 6

Multiplexer port for the left rear photodiode.

Definition at line 23 of file [CD4051.h](#).

Referenced by [CLS15_Update_Photodiode_Offset_Left_Rear\(\)](#), and [CLS15_Update_Photodiode_Value_Left\(\)](#).

3.28.2.5 #define Photodiode_rightfront 2

Multiplexer port for the right front photodiode.

Definition at line 17 of file [CD4051.h](#).

Referenced by [CLS15_Update_Photodiode_Offset_Right_Front\(\)](#), and [CLS15_Update_Photodiode_Value_Right\(\)](#).

3.28.2.6 #define Photodiode_rightrear 3

Multiplexer port for the right rear photodiode.

Definition at line 19 of file [CD4051.h](#).

Referenced by [CLS15_Update_Photodiode_Offset_Right_Rear\(\)](#), and [CLS15_Update_Photodiode_Value_Right\(\)](#).

3.28.2.7 #define Pressure_Sensor_Mux_Port 7

The default port for the pressure sensor on the ATmega328P.

Definition at line 25 of file [CD4051.h](#).

Referenced by [MPX5010GP_Read_Value\(\)](#), and [Test_Pressure_Sensor\(\)](#).

3.28.3 Function Documentation

3.28.3.1 void CD4051_Initialize (void)

This function is used to initialize the pins used for the multiplexer.

Definition at line 24 of file [CD4051.c](#).

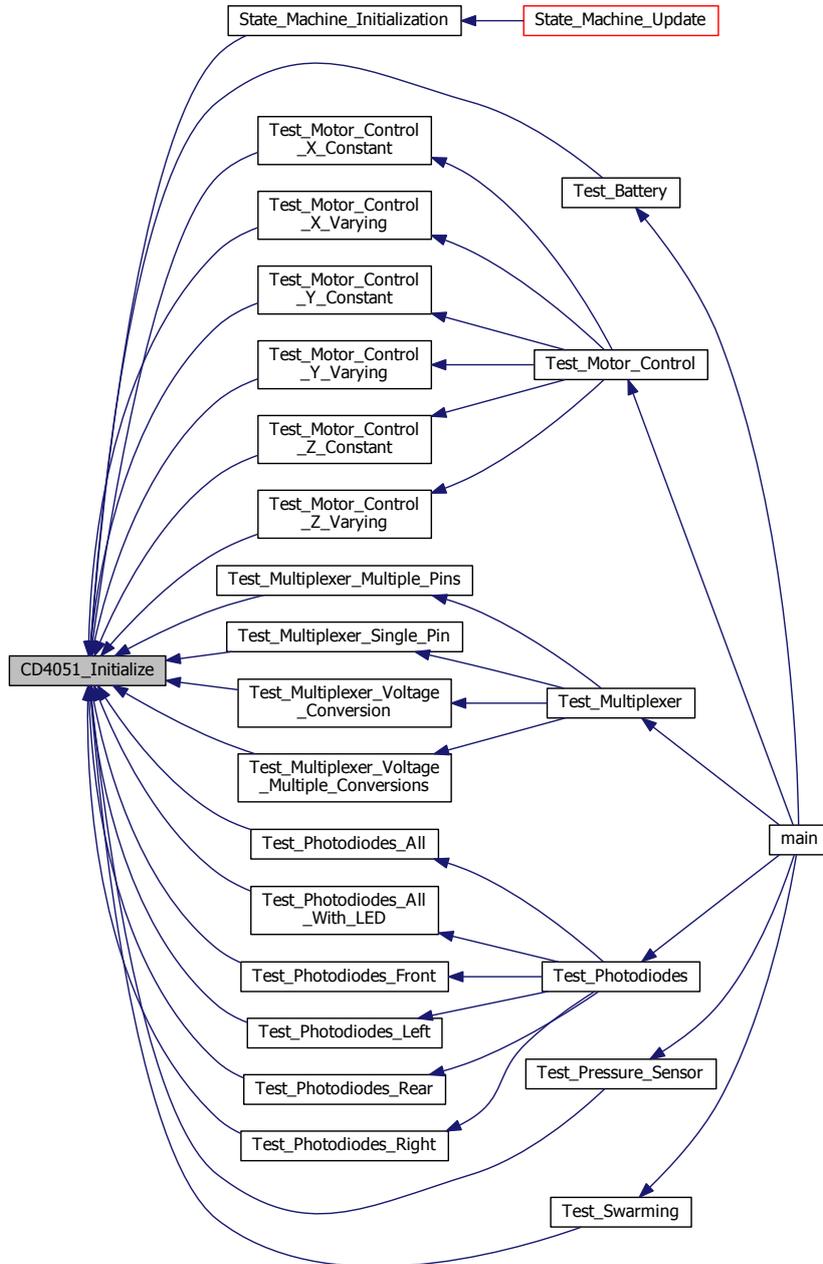
References [ADC_DEFAULT_PIN](#), and [ADC_Setup_Pin\(\)](#).

Referenced by [State_Machine_Initialization\(\)](#), [Test_Battery\(\)](#), [Test_Motor_Control_X_Constant\(\)](#), [Test_Motor_Control_X_Varying\(\)](#), [Test_Motor_Control_Y_Constant\(\)](#), [Test_Motor_Control_Y_Varying\(\)](#), [Test_Motor_Control_Z_Constant\(\)](#), [Test_Motor_Control_Z_Varying\(\)](#), [Test_Multiplexer_Multiple_Pins\(\)](#), [Test_Multiplexer_Single_Pin\(\)](#), [Test_Multiplexer_Voltage_Conversion\(\)](#), [Test_Multiplexer_Voltage_Multiple_Conversions\(\)](#), [Test_Photodiodes_All\(\)](#), [Test_Photodiodes_All_With_LED\(\)](#), [Test_Photodiodes_Front\(\)](#), [Test_Photodiodes_Left\(\)](#), [Test_Photodiodes_Rear\(\)](#), [Test_Photodiodes_Right\(\)](#), [Test_Pressure_Sensor\(\)](#), and [Test_Swarming\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.28.3.2 void CD4051_Read_Range (int * Output, unsigned char Minimum_Multiplexer_Pin, unsigned char Maximum_Multiplexer_Pin)

This function is used to calculate the distance for a photodiode.

Parameters

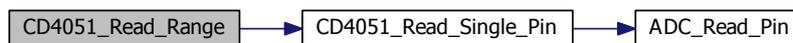
<i>Output</i>	An array holding ADC values for the selected multiplexer pins.
<i>Minimum_↔ Multiplexer_Pin</i>	Lowest pin range to be selected from the multiplexer.
<i>Maximum_↔ Multiplexer_Pin</i>	Highest pin range to be selected from the multiplexer.

Definition at line 64 of file [CD4051.c](#).

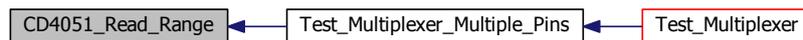
References [CD4051_Read_Single_Pin\(\)](#).

Referenced by [Test_Multiplexer_Multiple_Pins\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.28.3.3 int CD4051_Read_Single_Pin (unsigned char *Multiplexer_Pin*)

This function is to read the selected multiplexer pin Pin 0: Voltage

Pin 1: Photodiode Front

Pin 2: Photodiode Right Front

Pin 3: Photodiode Right Rear

Pin 4: Photodiode Left Front

Pin 5: Photodiode Left Rear

Pin 6: Photodiode Rear

Pin 7: Pressure Sensor

.

Parameters

<i>Multiplexer_Pin</i>	Pin to be selected from the multiplexer.
------------------------	--

Returns

ADC value for the selected multiplexer pin.

Warning

This function does not check the input for validity.

Definition at line 46 of file [CD4051.c](#).

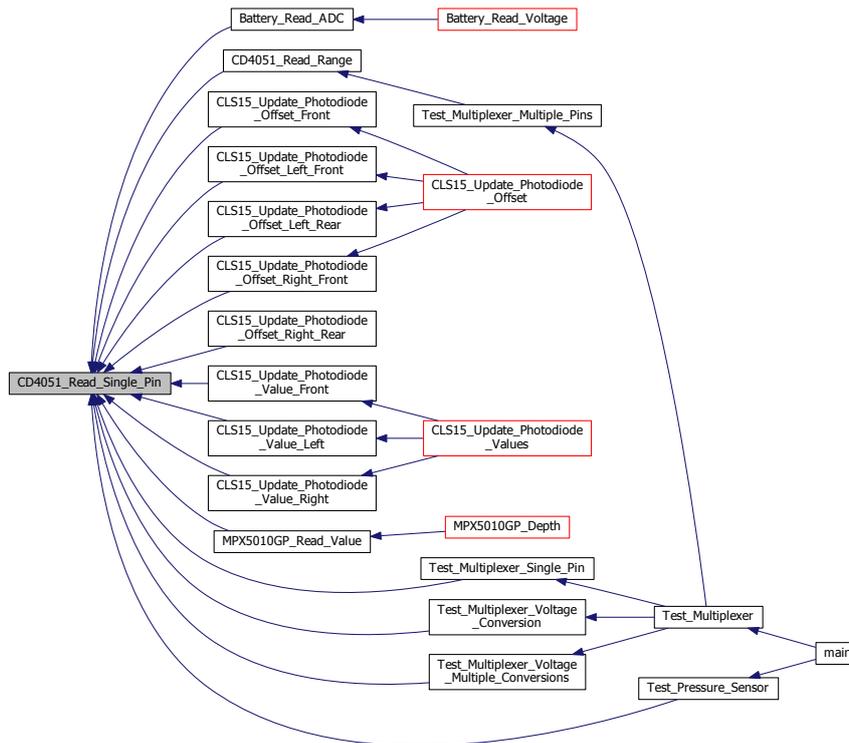
References [ADC_Read_Pin\(\)](#).

Referenced by [Battery_Read_ADC\(\)](#), [CD4051_Read_Range\(\)](#), [CLS15_Update_Photodiode_Offset_Front\(\)](#), [CLS15_Update_Photodiode_Offset_Left_Front\(\)](#), [CLS15_Update_Photodiode_Offset_Left_Rear\(\)](#), [CLS15_Update_Photodiode_Offset_Right_Front\(\)](#), [CLS15_Update_Photodiode_Offset_Right_Rear\(\)](#), [CLS15_Update_Photodiode_Value_Front\(\)](#), [CLS15_Update_Photodiode_Value_Left\(\)](#), [CLS15_Update_Photodiode_Value_Right\(\)](#), [MPX5010GP_Read_Value\(\)](#), [Test_Multiplexer_Single_Pin\(\)](#), [Test_Multiplexer_Voltage_Conversion\(\)](#), [Test_Multiplexer_Voltage_Multiple_Conversions\(\)](#), and [Test_Pressure_Sensor\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.29 CD4051.h

```

00001 /**
00002  * @file CD4051.h
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 1/22/2015\n Created: 10/23/2014 3:23:07 PM
00006  * @brief This file is code to control the CD4051 multiplexer connected to the ADC.
00007  */
00008
00009 #ifndef CD4051_H_
  
```

```

00010 #define CD4051_H_
00011
00012 /** @brief The port on the multiplexer that has the current reference voltage. */
00013 #define Battery_Mux_Port 0
00014 /** @brief Multiplexer port for the front photodiode. */
00015 #define Photodiode_front 1
00016 /** @brief Multiplexer port for the right front photodiode. */
00017 #define Photodiode_rightfront 2
00018 /** @brief Multiplexer port for the right rear photodiode. */
00019 #define Photodiode_rightrear 3
00020 /** @brief Multiplexer port for the left front photodiode. */
00021 #define Photodiode_leftfront 4
00022 /** @brief Multiplexer port for the left rear photodiode. */
00023 #define Photodiode_leftrear 6
00024 /** @brief The default port for the pressure sensor on the ATmega328P. */
00025 #define Pressure_Sensor_Mux_Port 7
00026
00027 void CD4051_Initialize(void);
00028 void CD4051_Read_Range(int *Output, unsigned char Minimum_Multiplexer_Pin, unsigned char
Maximum_Multiplexer_Pin);
00029 int CD4051_Read_Single_Pin(unsigned char Multiplexer_Pin);
00030
00031 #endif /* CD4051_H_ */

```

3.30 Microcontroller/Communication/I2C_Interface.c File Reference

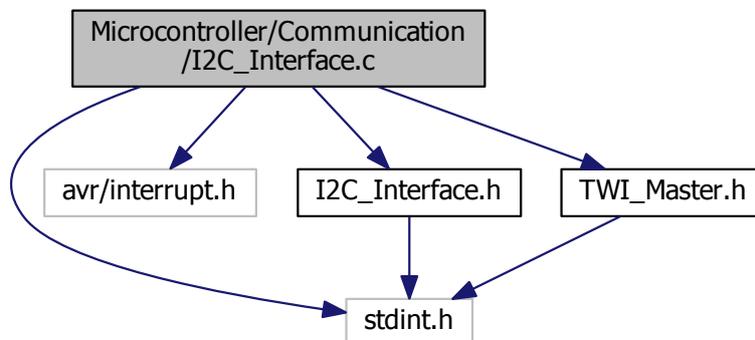
This file is code to control the TWI_Master.

```

#include <stdint.h>
#include <avr/interrupt.h>
#include "I2C_Interface.h"
#include "TWI_Master.h"

```

Include dependency graph for I2C_Interface.c:



Macros

- #define `F_CPU` 16000000UL
The current clock speed for the microcontroller.

Functions

- void `I2C_Interface_Initialize` ()
This function is to initialize the I2C interface.
- void `I2C_Interface_Write` (uint8_t Device_Address, uint8_t Device_Register, uint8_t Value_To_Write)

This function is to send a write I2C message over the I2C bus.

- `uint8_t I2C_Interface_Single_Read` (`uint8_t Device_Address`, `uint8_t Device_Register`)

This function is to send and receive a read I2C message over the I2C bus for a single read.

- `void I2C_Interface_Read_Array` (`uint8_t Device_Address`, `uint8_t Device_Register`, `uint8_t *Return_Buff`, `uint8_t Size_Of_Return_Buff`)

This function is to send and receive a read I2C message over the I2C bus for multiple reads.

- `uint8_t I2C_Interface_Not_Busy` ()

This function is to check if the I2C interface is not busy.

3.30.1 Detailed Description

This file is code to control the TWI_Master.

Author

Nicholas Sikkema

Version

Revision: 1.5

Date

Last Updated: 2/3/2015

Created: 10/30/2014 3:47:02 PM

Definition in file [I2C_Interface.c](#).

3.30.2 Macro Definition Documentation

3.30.2.1 #define F_CPU 16000000UL

The current clock speed for the microcontroller.

Definition at line 11 of file [I2C_Interface.c](#).

3.30.3 Function Documentation

3.30.3.1 void I2C_Interface_Initialize (void)

This function is to initialize the I2C interface.

Definition at line 22 of file [I2C_Interface.c](#).

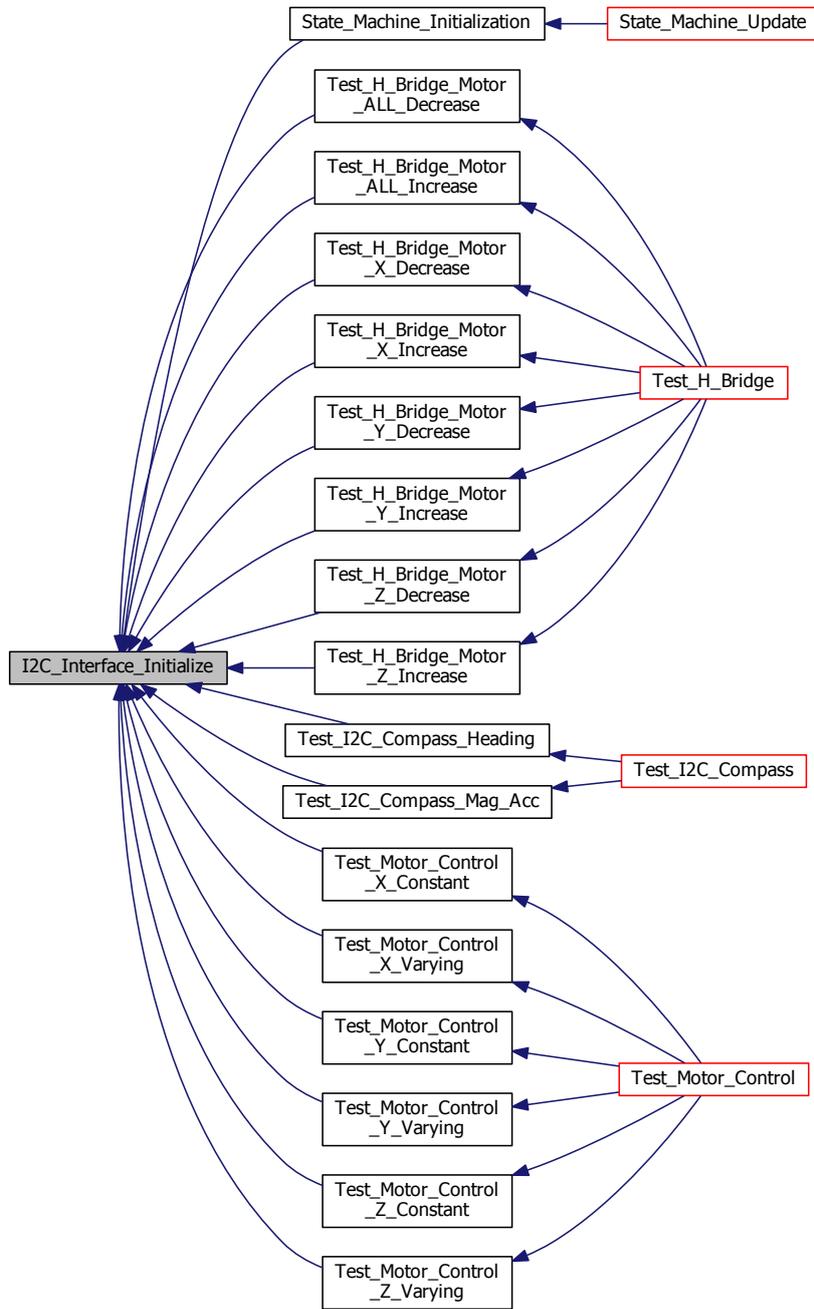
References [TWI_Master_Initialise\(\)](#).

Referenced by [State_Machine_Initialization\(\)](#), [Test_H_Bridge_Motor_ALL_Decrease\(\)](#), [Test_H_Bridge_Motor_AL↔L_Increase\(\)](#), [Test_H_Bridge_Motor_X_Decrease\(\)](#), [Test_H_Bridge_Motor_X_Increase\(\)](#), [Test_H_Bridge_Motor↔_Y_Decrease\(\)](#), [Test_H_Bridge_Motor_Y_Increase\(\)](#), [Test_H_Bridge_Motor_Z_Decrease\(\)](#), [Test_H_Bridge↔_Motor_Z_Increase\(\)](#), [Test_I2C_Compass_Heading\(\)](#), [Test_I2C_Compass_Mag_Acc\(\)](#), [Test_Motor_Control↔_X_Constant\(\)](#), [Test_Motor_Control_X_Varying\(\)](#), [Test_Motor_Control_Y_Constant\(\)](#), [Test_Motor_Control_Y↔_Varying\(\)](#), [Test_Motor_Control_Z_Constant\(\)](#), and [Test_Motor_Control_Z_Varying\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.30.3.2 uint8_t I2C_Interface_Not_Busy (void)

This function is to check if the I2C interface is not busy.

Returns

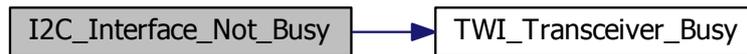
Boolean true (1) or false (0) for not being busy.

Definition at line 103 of file [I2C_Interface.c](#).

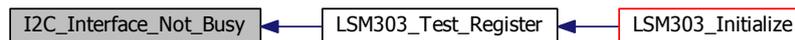
References [TWI_Transceiver_Busy\(\)](#).

Referenced by [LSM303_Test_Register\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.30.3.3 `void I2C_Interface_Read_Array (uint8_t Device_Address, uint8_t Device_Register, uint8_t * Return_Buff, uint8_t Size_Of_Return_Buff)`

This function is to send and receive a read I2C message over the I2C bus for multiple reads.

Parameters

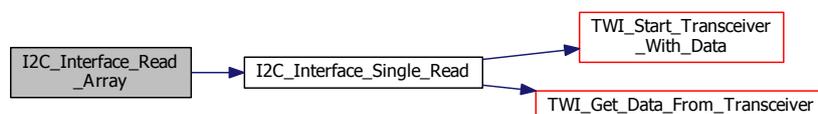
<i>Device_Address</i>	The I2C device address.
<i>Device_Register</i>	The register to read the data. Note that this function expects that the registers that you are reading from a continuous.
<i>Return_Buff</i>	An array to store the data from the I2C bus.
<i>Size_Of_Return_Buff</i>	Size of the return buffer array.

Definition at line 87 of file [I2C_Interface.c](#).

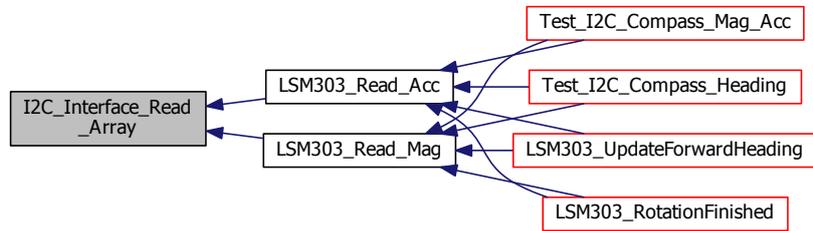
References [I2C_Interface_Single_Read\(\)](#).

Referenced by [LSM303_Read_Acc\(\)](#), and [LSM303_Read_Mag\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.30.3.4 uint8_t I2C_Interface_Single_Read (uint8_t Device_Address, uint8_t Device_Register)

This function is to send and receive a read I2C message over the I2C bus for a single read.

Parameters

<i>Device_Address</i>	The I2C device address.
<i>Device_Register</i>	The register to read the data.

Returns

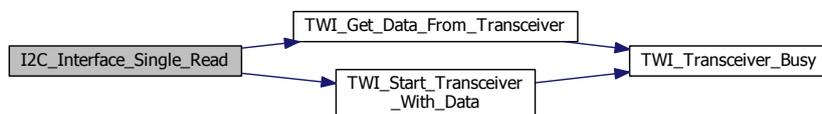
The value from the register wanted.

Definition at line 56 of file [I2C_Interface.c](#).

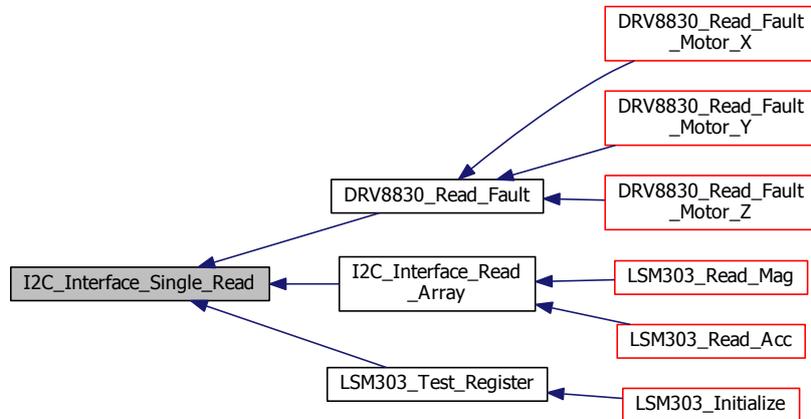
References [FALSE](#), [TRUE](#), [TWI_Get_Data_From_Transceiver\(\)](#), [TWI_READ_BIT](#), and [TWI_Start_Transceiver_With_Data\(\)](#).

Referenced by [DRV8830_Read_Fault\(\)](#), [I2C_Interface_Read_Array\(\)](#), and [LSM303_Test_Register\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.30.3.5 void I2C_Interface_Write (uint8_t Device_Address, uint8_t Device_Register, uint8_t Value_To_Write)

This function is to send a write I2C message over the I2C bus.

Parameters

<i>Device_Address</i>	The I2C device address.
<i>Device_Register</i>	The register to write the data.
<i>Value_To_Write</i>	Data to be sent over the bus.

Definition at line 36 of file [I2C_Interface.c](#).

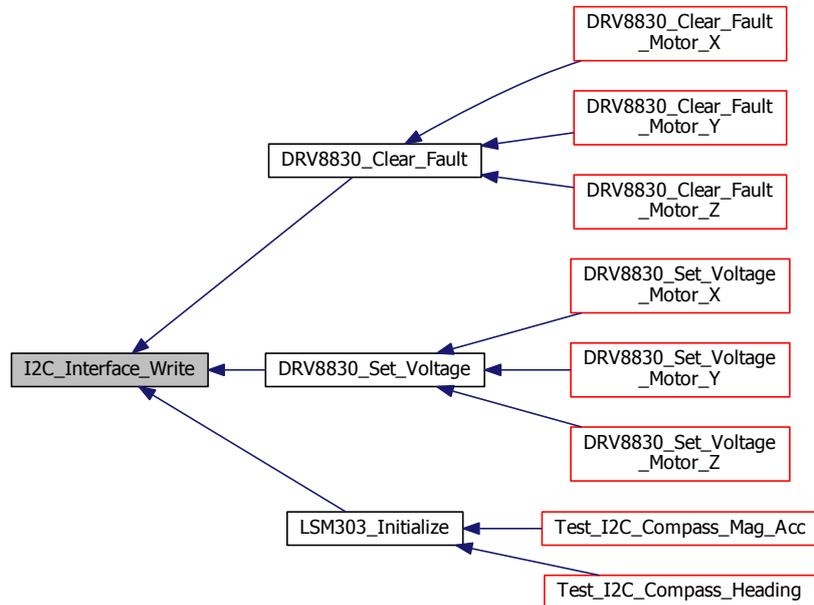
References [FALSE](#), [TWI_READ_BIT](#), and [TWI_Start_Transceiver_With_Data\(\)](#).

Referenced by [DRV8830_Clear_Fault\(\)](#), [DRV8830_Set_Voltage\(\)](#), and [LSM303_Initialize\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.31 I2C_Interface.c

```

00001 /**
00002  * @file I2C_Interface.c
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.5
00005  * @date Last Updated: 2/3/2015\n Created: 10/30/2014 3:47:02 PM
00006  * @brief This file is code to control the TWI_Master.
00007  */
00008
00009 #ifndef F_CPU
00010     /** @brief The current clock speed for the microcontroller. */
00011     #define F_CPU 16000000UL
00012 #endif
00013
00014 #include <stdint.h>
00015 #include <avr/interrupt.h>
00016 #include "I2C_Interface.h"
00017 #include "TWI_Master.h"
00018
00019 /**
00020  * @brief This function is to initialize the I2C interface.
00021  */
00022 void I2C_Interface_Initialize()
00023 {
00024     /* Initialize the TWI master. */
00025     TWI_Master_Initialise();
00026     /* Enable global interrupts. */
00027     sei();
00028 }
00029
00030 /**
00031  * @brief This function is to send a write I2C message over the I2C bus.
00032  * @param Device_Address The I2C device address.
00033  * @param Device_Register The register to write the data.
00034  * @param Value_To_Write Data to be sent over the bus.
00035  */
00036 void I2C_Interface_Write(uint8_t Device_Address, uint8_t Device_Register, uint8_t
Value_To_Write)
00037 {
00038     /* Initialize the message buffer. */
00039     uint8_t Message_Buff[3];
00040     /* Set array index 0 to the device address (bits 1-7) and the TWI_READ_BIT in write mode (bit 0). */
00041     Message_Buff[0] = (Device_Address<<1)|(FALSE<<TWI_READ_BIT);
  
```

```

00042     /* Set array index 1 to the device register you want to write to. */
00043     Message_Buff[1] = Device_Register;
00044     /* Set array index 2 to the value you want to write. */
00045     Message_Buff[2] = Value_To_Write;
00046     /* Send the write message over the I2C bus. */
00047     TWI_Start_Transceiver_With_Data(Message_Buff, 3);
00048 }
00049
00050 /**
00051  * @brief This function is to send and receive a read I2C message over the I2C bus for a single read.
00052  * @param Device_Address The I2C device address.
00053  * @param Device_Register The register to read the data.
00054  * @return The value from the register wanted.
00055  */
00056 uint8_t I2C_Interface_Single_Read(uint8_t Device_Address, uint8_t Device_Register)
00057 {
00058     /* Initialize a buffer to hold the information from the I2C bus. */
00059     uint8_t Message_Buff[2];
00060     /* Set array index 0 to the device address (bits 1-7) and the TWI_READ_BIT in write mode (bit 0). */
00061     Message_Buff[0] = (Device_Address<<1)|(FALSE<<TWI_READ_BIT);
00062     /* Set array index 1 to the device register you want to read from. */
00063     Message_Buff[1] = Device_Register;
00064     /* Send the I2C message to indicate specific location and sub. */
00065     TWI_Start_Transceiver_With_Data(Message_Buff, 2);
00066     /* Set array index 0 to the device address (bits 1-7) and the TWI_READ_BIT in read mode (bit 0). */
00067     Message_Buff[0] = (Device_Address<<1)|(TRUE<<TWI_READ_BIT);
00068     /* Send a dummy write to set the read pointer for the device. */
00069     TWI_Start_Transceiver_With_Data(Message_Buff, 2);
00070     /* Clear the message buffer. */
00071     Message_Buff[0] = 0;
00072     Message_Buff[1] = 0;
00073     /* Retrieve the data from the I2C bus. */
00074     TWI_Get_Data_From_Transceiver(Message_Buff, 2);
00075     /* Set the return buffer with the value from the I2C bus. */
00076     return(Message_Buff[1]);
00077 }
00078
00079 /**
00080  * @brief This function is to send and receive a read I2C message over the I2C bus for multiple reads.
00081  * @param Device_Address The I2C device address.
00082  * @param Device_Register The register to read the data.\n
00083  * Note that this function expects that the registers that you are reading from a continuous.
00084  * @param Return_Buff An array to store the data from the I2C bus.
00085  * @param Size_Of_Return_Buff Size of the return buffer array.
00086  */
00087 void I2C_Interface_Read_Array(uint8_t Device_Address, uint8_t Device_Register,
00088     uint8_t *Return_Buff, uint8_t Size_Of_Return_Buff)
00089 {
00089     /* Loop for the indexes. */
00090     for(int i=0; i<Size_Of_Return_Buff; i++)
00091     {
00092         /* Clear the return buffer. */
00093         Return_Buff[i] = 0;
00094         /* Set the return buffer with the value from the I2C bus. */
00095         Return_Buff[i] = I2C_Interface_Single_Read(Device_Address, Device_Register
00096     +i);
00097     }
00098 }
00099 /**
00100  * @brief This function is to check if the I2C interface is not busy.
00101  * @return Boolean true (1) or false (0) for not being busy.
00102  */
00103 uint8_t I2C_Interface_Not_Busy()
00104 {
00105     /* Return the opposite of TWI_Transceiver_Busy(). */
00106     return(!TWI_Transceiver_Busy());
00107 }

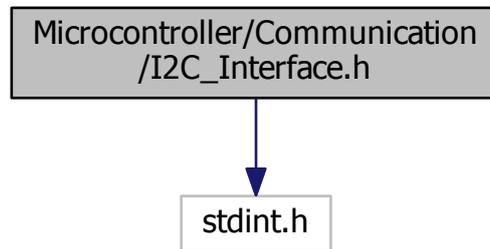
```

3.32 Microcontroller/Communication/I2C_Interface.h File Reference

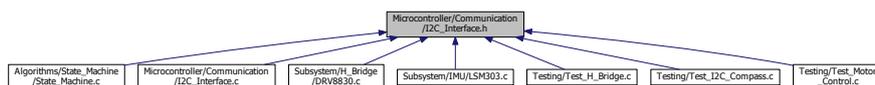
This file is code to control the TWI_Master.

```
#include <stdint.h>
```

Include dependency graph for I2C_Interface.h:



This graph shows which files directly or indirectly include this file:



Functions

- void [I2C_Interface_Initialize](#) (void)
This function is to initialize the I2C interface.
- void [I2C_Interface_Write](#) (uint8_t Device_Address, uint8_t Device_Register, uint8_t Value_To_Write)
This function is to send a write I2C message over the I2C bus.
- uint8_t [I2C_Interface_Single_Read](#) (uint8_t Device_Address, uint8_t Device_Register)
This function is to send and receive a read I2C message over the I2C bus for a single read.
- void [I2C_Interface_Read_Array](#) (uint8_t Device_Address, uint8_t Device_Register, uint8_t *Return_Buff, uint8_t Size_Of_Return_Buff)
This function is to send and receive a read I2C message over the I2C bus for multiple reads.
- uint8_t [I2C_Interface_Not_Busy](#) (void)
This function is to check if the I2C interface is not busy.

3.32.1 Detailed Description

This file is code to control the TWI_Master.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 1/22/2015

Created: 10/30/2014 4:09:09 PM

Definition in file [I2C_Interface.h](#).

3.32.2 Function Documentation

3.32.2.1 void I2C_Interface_Initialize (void)

This function is to initialize the I2C interface.

Definition at line 22 of file [I2C_Interface.c](#).

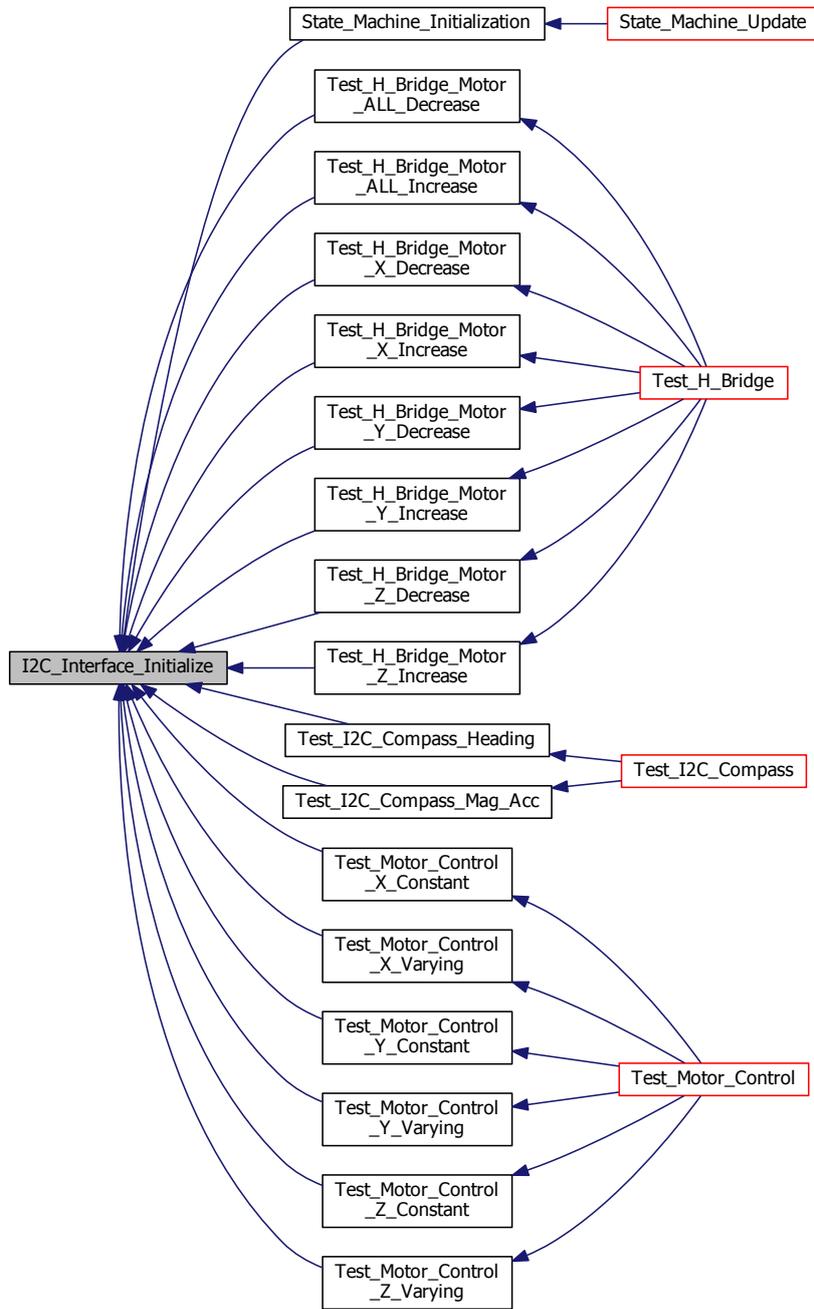
References [TWI_Master_Initialise\(\)](#).

Referenced by [State_Machine_Initialization\(\)](#), [Test_H_Bridge_Motor_ALL_Decrease\(\)](#), [Test_H_Bridge_Motor_AL↔L_Increase\(\)](#), [Test_H_Bridge_Motor_X_Decrease\(\)](#), [Test_H_Bridge_Motor_X_Increase\(\)](#), [Test_H_Bridge_Motor↔_Y_Decrease\(\)](#), [Test_H_Bridge_Motor_Y_Increase\(\)](#), [Test_H_Bridge_Motor_Z_Decrease\(\)](#), [Test_H_Bridge↔_Motor_Z_Increase\(\)](#), [Test_I2C_Compass_Heading\(\)](#), [Test_I2C_Compass_Mag_Acc\(\)](#), [Test_Motor_Control↔_X_Constant\(\)](#), [Test_Motor_Control_X_Varying\(\)](#), [Test_Motor_Control_Y_Constant\(\)](#), [Test_Motor_Control_Y↔_Varying\(\)](#), [Test_Motor_Control_Z_Constant\(\)](#), and [Test_Motor_Control_Z_Varying\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.32.2.2 uint8_t I2C_Interface_Not_Busy (void)

This function is to check if the I2C interface is not busy.

Returns

Boolean true (1) or false (0) for not being busy.

Definition at line 103 of file [I2C_Interface.c](#).

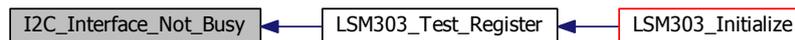
References [TWI_Transceiver_Busy\(\)](#).

Referenced by [LSM303_Test_Register\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.32.2.3 `void I2C_Interface_Read_Array (uint8_t Device_Address, uint8_t Device_Register, uint8_t * Return_Buff, uint8_t Size_Of_Return_Buff)`

This function is to send and receive a read I2C message over the I2C bus for multiple reads.

Parameters

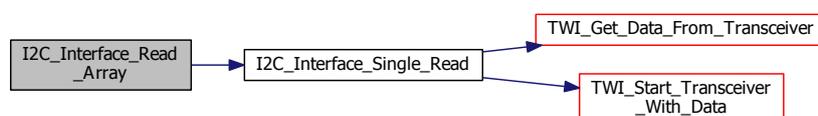
<i>Device_Address</i>	The I2C device address.
<i>Device_Register</i>	The register to read the data. Note that this function expects that the registers that you are reading from a continuous.
<i>Return_Buff</i>	An array to store the data from the I2C bus.
<i>Size_Of_Return_Buff</i>	Size of the return buffer array.

Definition at line 87 of file [I2C_Interface.c](#).

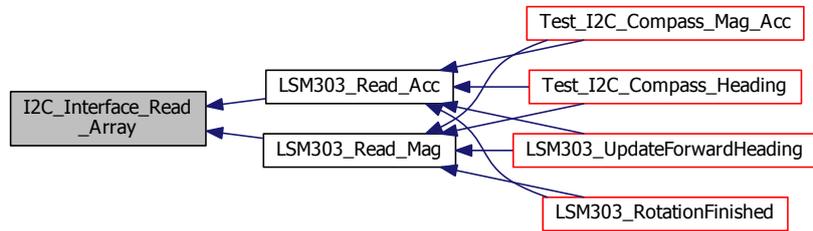
References [I2C_Interface_Single_Read\(\)](#).

Referenced by [LSM303_Read_Acc\(\)](#), and [LSM303_Read_Mag\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.32.2.4 uint8_t I2C_Interface_Single_Read (uint8_t Device_Address, uint8_t Device_Register)

This function is to send and receive a read I2C message over the I2C bus for a single read.

Parameters

<i>Device_Address</i>	The I2C device address.
<i>Device_Register</i>	The register to read the data.

Returns

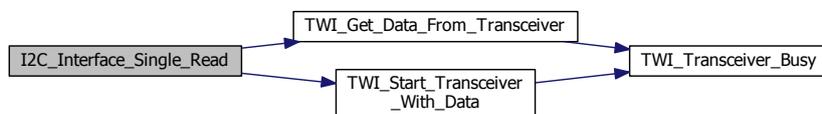
The value from the register wanted.

Definition at line 56 of file [I2C_Interface.c](#).

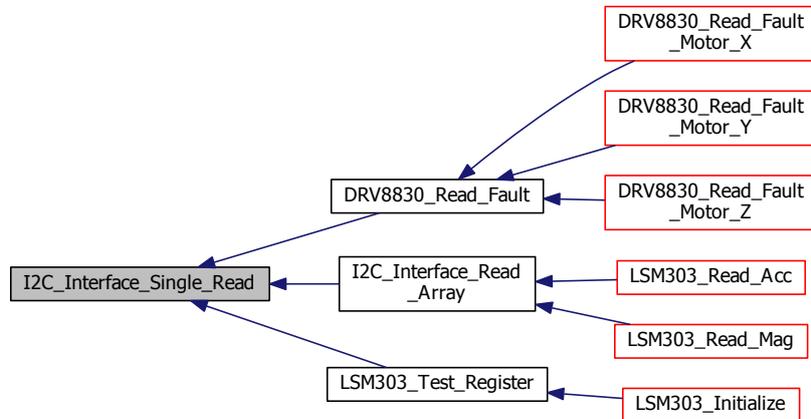
References [FALSE](#), [TRUE](#), [TWI_Get_Data_From_Transceiver\(\)](#), [TWI_READ_BIT](#), and [TWI_Start_Transceiver_With_Data\(\)](#).

Referenced by [DRV8830_Read_Fault\(\)](#), [I2C_Interface_Read_Array\(\)](#), and [LSM303_Test_Register\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.32.2.5 void I2C_Interface_Write (uint8_t Device_Address, uint8_t Device_Register, uint8_t Value_To_Write)

This function is to send a write I2C message over the I2C bus.

Parameters

<i>Device_Address</i>	The I2C device address.
<i>Device_Register</i>	The register to write the data.
<i>Value_To_Write</i>	Data to be sent over the bus.

Definition at line 36 of file [I2C_Interface.c](#).

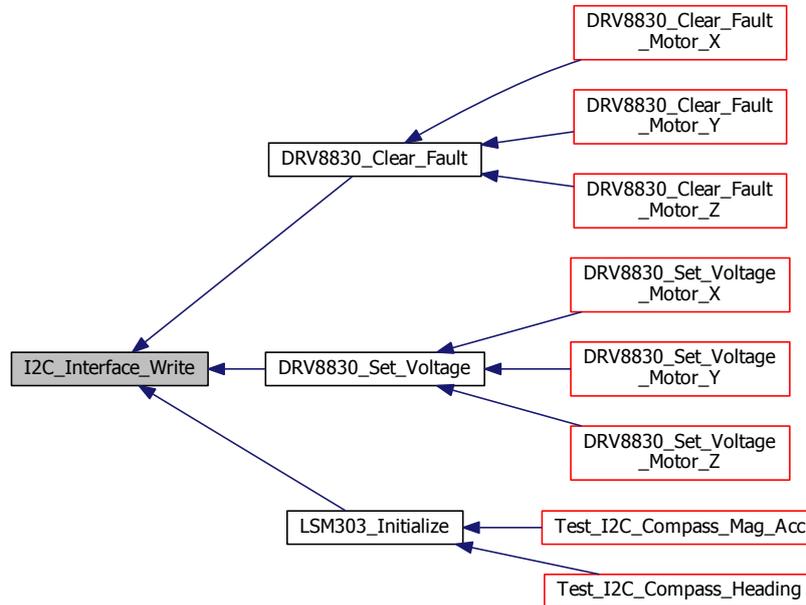
References [FALSE](#), [TWI_READ_BIT](#), and [TWI_Start_Transceiver_With_Data\(\)](#).

Referenced by [DRV8830_Clear_Fault\(\)](#), [DRV8830_Set_Voltage\(\)](#), and [LSM303_Initialize\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.33 I2C_Interface.h

```

00001 /**
00002  * @file I2C_Interface.h
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 1/22/2015\n Created: 10/30/2014 4:09:09 PM
00006  * @brief This file is code to control the TWI_Master.
00007  */
00008
00009 #ifndef I2C_INTERFACE_H_
00010 #define I2C_INTERFACE_H_
00011
00012 #include <stdint.h>
00013
00014 void I2C_Interface_Initialize(void);
00015 void I2C_Interface_Write(uint8_t Device_Address, uint8_t Device_Register, uint8_t
Value_To_Write);
00016 uint8_t I2C_Interface_Single_Read(uint8_t Device_Address, uint8_t Device_Register)
;
00017 void I2C_Interface_Read_Array(uint8_t Device_Address, uint8_t Device_Register,
uint8_t* Return_Buff, uint8_t Size_Of_Return_Buff);
00018 uint8_t I2C_Interface_Not_Busy(void);
00019
00020 #endif /* I2C_INTERFACE_H_ */
  
```

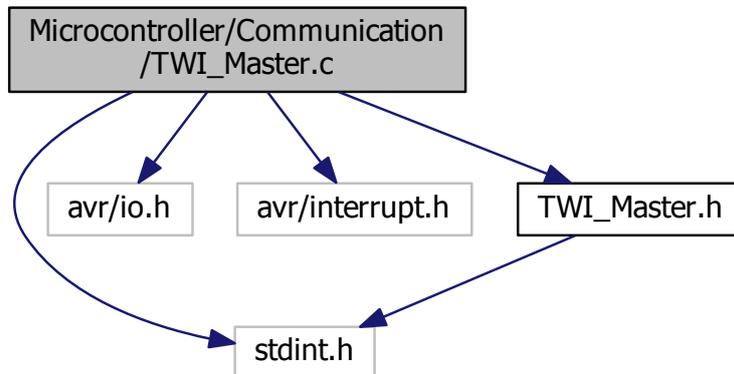
3.34 Microcontroller/Communication/TWI_Master.c File Reference

This is a sample driver for the TWI hardware modules. It is interrupt driven. All functionality is controlled through passing information to and from functions.

```

#include <stdint.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include "TWI_Master.h"
  
```

Include dependency graph for TWI_Master.c:



Functions

- void [TWI_Master_Initialise](#) (void)

Call this function to set up the TWI master to its initial standby state.
- uint8_t [TWI_Transceiver_Busy](#) (void)

Call this function to test if the TWI_ISR is busy transmitting.
- uint8_t [TWI_Get_State_Info](#) (void)

Call this function to fetch the state information of the previous operation. The function will hold execution (loop) until the TWI_ISR has completed with the previous operation.
- void [TWI_Start_Transceiver_With_Data](#) (uint8_t *msg, uint8_t msgSize)

Call this function to send a prepared message. The first byte must contain the slave address and the read/write bit. Consecutive bytes contain the data to be sent, or empty locations for data to be read from the slave. Also include how many bytes that should be sent/read including the address byte. The function will hold execution (loop) until the TWI_ISR has completed with the previous operation, then initialize the next operation and return.
- void [TWI_Start_Transceiver](#) (void)

Call this function to resend the last message. The driver will reuse the data previously put in the transceiver buffers. The function will hold execution (loop) until the TWI_ISR has completed with the previous operation, then initialize the next operation and return.
- uint8_t [TWI_Get_Data_From_Transceiver](#) (uint8_t *msg, uint8_t msgSize)

Call this function to read out the requested data from the TWI transceiver buffer. I.e. first call [TWI_Start_Transceiver](#) to send a request for data to the slave. Then Run this function to collect the data when they have arrived. Include a pointer to where to place the data and the number of bytes requested (including the address field) in the function call. The function will hold execution (loop) until the TWI_ISR has completed with the previous operation, before reading out the data and returning.
- [ISR](#) (TWI_vect)

This function is the Interrupt Service Routine (ISR), and called when the TWI interrupt is triggered; that is whenever a TWI event has occurred. This function should not be called directly from the main application.

3.34.1 Detailed Description

This is a sample driver for the TWI hardware modules. It is interrupt driven. All functionality is controlled through passing information to and from functions.

Atmel Corporation

Author

Itwa Commented by: Nicholas Sikkema

Version

Revision: 1.13

Date

24. mai 2004 11:31:20

Definition in file [TWI_Master.c](#).

3.34.2 Function Documentation**3.34.2.1 ISR (TWI_vect)**

This function is the Interrupt Service Routine (ISR), and called when the TWI interrupt is triggered; that is whenever a TWI event has occurred. This function should not be called directly from the main application.

Definition at line [156](#) of file [TWI_Master.c](#).

References [TRUE](#), [TWI_ARB_LOST](#), [TWI_BUS_ERROR](#), [TWI_MRX_ADR_ACK](#), [TWI_MRX_ADR_NACK](#), [TWI_MRX_DATA_ACK](#), [TWI_MRX_DATA_NACK](#), [TWI_MTX_ADR_ACK](#), [TWI_MTX_ADR_NACK](#), [TWI_MTX_DATA_ACK](#), [TWI_MTX_DATA_NACK](#), [TWI_REP_START](#), and [TWI_START](#).

3.34.2.2 uint8_t TWI_Get_Data_From_Transceiver (uint8_t * msg, uint8_t msgSize)

Call this function to read out the requested data from the TWI transceiver buffer. I.e. first call [TWI_Start_Transceiver](#) to send a request for data to the slave. Then Run this function to collect the data when they have arrived. Include a pointer to where to place the data and the number of bytes requested (including the address field) in the function call. The function will hold execution (loop) until the [TWI_ISR](#) has completed with the previous operation, before reading out the data and returning.

Parameters

<i>msg</i>	Message buffer to be received from the I2C bus.
<i>msgSize</i>	Size of the message buffer to be received from the I2C bus.

Returns

Definition at line [133](#) of file [TWI_Master.c](#).

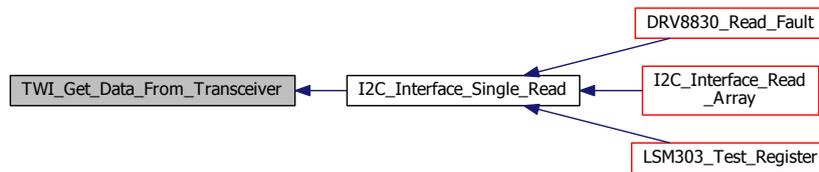
References [TWI_Transceiver_Busy\(\)](#).

Referenced by [I2C_Interface_Single_Read\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.34.2.3 `uint8_t TWI_Get_State_Info (void)`

Call this function to fetch the state information of the previous operation. The function will hold execution (loop) until the `TWI_ISR` has completed with the previous operation.

Returns

The TWI state code if there was an error.

Definition at line 61 of file [TWI_Master.c](#).

References [TWI_Transceiver_Busy\(\)](#).

Here is the call graph for this function:



3.34.2.4 `void TWI_Master_Initialise (void)`

Call this function to set up the TWI master to its initial standby state.

Warning

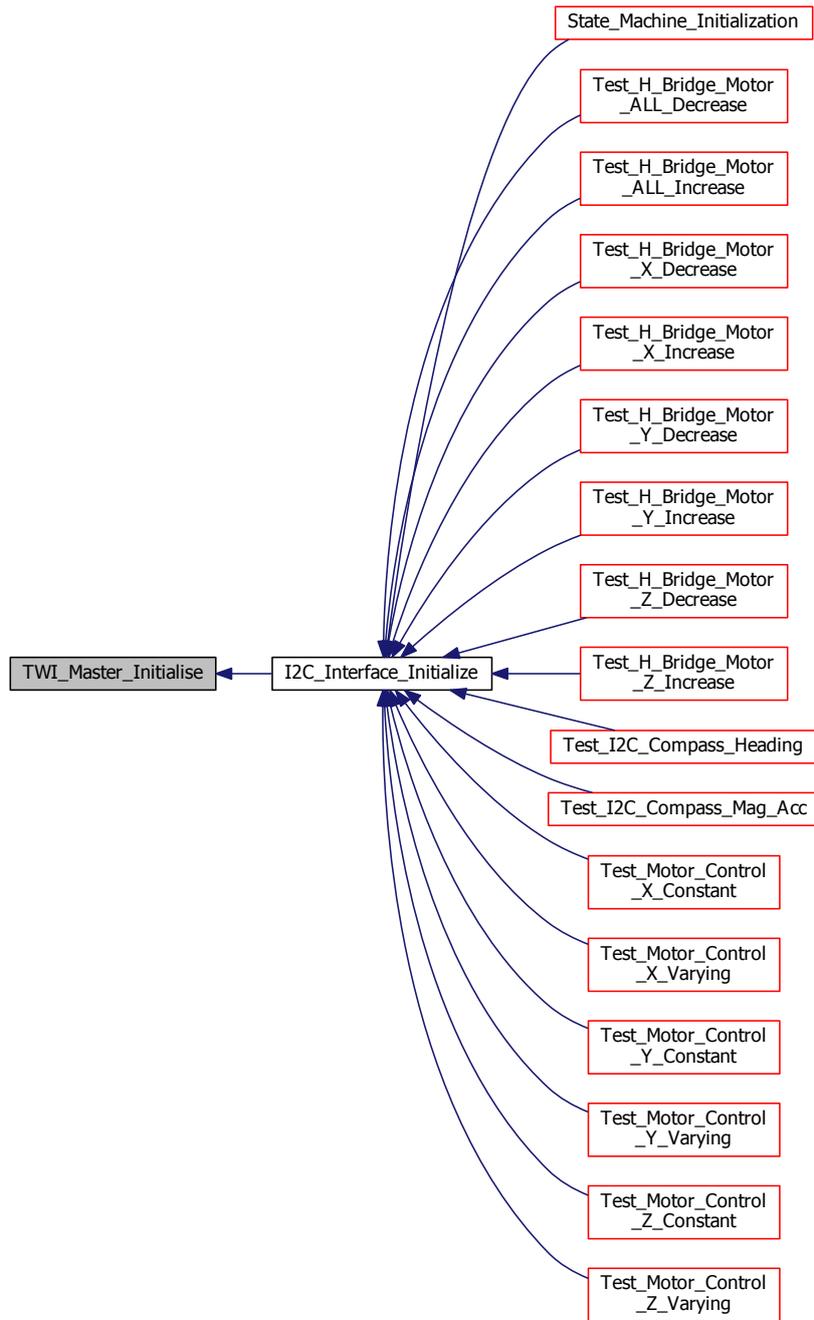
Remember to enable interrupts from the main application after initializing the TWI.

Definition at line 34 of file [TWI_Master.c](#).

References [TWI_TWBR](#), and [TWI_TWPS](#).

Referenced by [I2C_Interface_Initialize\(\)](#).

Here is the caller graph for this function:



3.34.2.5 void TWI_Start_Transceiver (void)

Call this function to resend the last message. The driver will reuse the data previously put in the transceiver buffers. The function will hold execution (loop) until the TWI_ISR has completed with the previous operation, then initialize the next operation and return.

Definition at line 111 of file [TWI_Master.c](#).

References [TWI_NO_STATE](#), and [TWI_Transceiver_Busy\(\)](#).

Here is the call graph for this function:



3.34.2.6 void TWI_Start_Transceiver_With_Data (uint8_t * msg, uint8_t msgSize)

Call this function to send a prepared message. The first byte must contain the slave address and the read/write bit. Consecutive bytes contain the data to be sent, or empty locations for data to be read from the slave. Also include how many bytes that should be sent/read including the address byte. The function will hold execution (loop) until the TWI_ISR has completed with the previous operation, then initialize the next operation and return.

Parameters

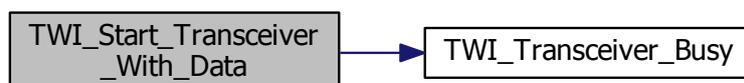
<i>msg</i>	Message buffer to be sent over the I2C bus.
<i>msgSize</i>	Size of the message buffer to be sent over the I2C bus.

Definition at line 78 of file [TWI_Master.c](#).

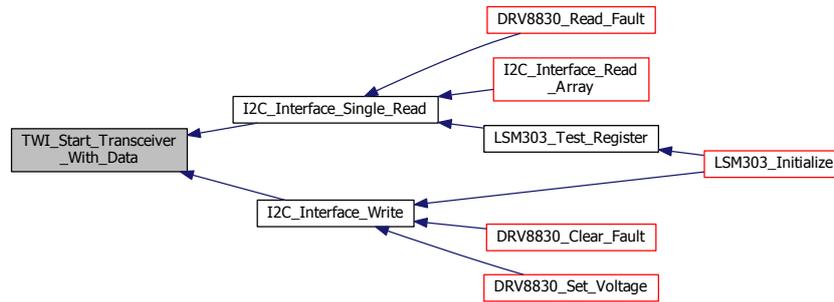
References [TRUE](#), [TWI_NO_STATE](#), [TWI_READ_BIT](#), and [TWI_Transceiver_Busy\(\)](#).

Referenced by [I2C_Interface_Single_Read\(\)](#), and [I2C_Interface_Write\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.34.2.7 uint8_t TWI_Transceiver_Busy (void)

Call this function to test if the TWI_ISR is busy transmitting.

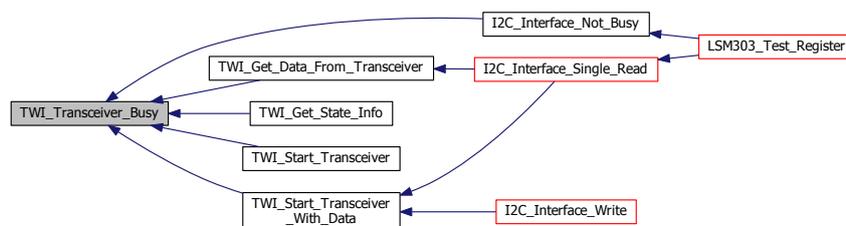
Returns

Status of the transceiver.

Definition at line 50 of file [TWI_Master.c](#).

Referenced by [I2C_Interface_Not_Busy\(\)](#), [TWI_Get_Data_From_Transceiver\(\)](#), [TWI_Get_State_Info\(\)](#), [TWI_Start_Transceiver\(\)](#), and [TWI_Start_Transceiver_With_Data\(\)](#).

Here is the caller graph for this function:



3.35 TWI_Master.c

```

00001 /**
00002  * Atmel Corporation
00003  * @file TWI_master.c
00004  * @author ltwa
00005  * Commented by: Nicholas Sikkema
00006  * @version Revision: 1.13
00007  * @date 24. mai 2004 11:31:20
00008  * @brief This is a sample driver for the TWI hardware modules.
00009  * It is interrupt driven. All functionality is controlled through
00010  * passing information to and from functions.
00011  */
00012
00013 #include <stdint.h>
00014 #include <avr/io.h>
00015 #include <avr/interrupt.h>
00016 #include "TWI_Master.h"
  
```

```

00017
00018 /* Transceiver buffer */
00019 static uint8_t TWI_buf[TWI_BUFFER_SIZE];
00020 /* Number of bytes to be transmitted. */
00021 static uint8_t TWI_msgSize;//
00022 /* TWI state byte. Default set to TWI_NO_STATE. */
00023 static uint8_t TWI_state = TWI_NO_STATE;//
00024
00025 /* Status byte holding flags. */
00026 static volatile uint8_t lastTransOK = 0;
00027 /* Register for the TWI status. */
00028 static volatile uint8_t TWI_statusReg = 0;
00029
00030 /**
00031  * @brief Call this function to set up the TWI master to its initial standby state.
00032  * @warning Remember to enable interrupts from the main application after initializing the TWI.
00033  */
00034 void TWI_Master_Initialise(void)
00035 {
00036     /* Set bit rate register (Baudrate). Defined in header file. */
00037     TWBR = TWI_TWBR;
00038     /* Set TWI prescaler. */
00039     TWSR = TWI_TWPS;
00040     /* Default content = SDA released. */
00041     TWDR = 0xFF;
00042     /* Enable TWI-interface and release TWI pins, Disable Interrupt, and No Signal requests. */
00043     TWCR = (1<<TWEN) | (0<<TWIE) | (0<<TWINT) | (0<<TWEA) | (0<<TWSTA) | (0<<TWSTO) | (0<<TWWC);
00044 }
00045
00046 /**
00047  * @brief Call this function to test if the TWI_ISR is busy transmitting.
00048  * @return Status of the transceiver.
00049  */
00050 uint8_t TWI_Transceiver_Busy(void)
00051 {
00052     /* If the TWI Interrupt is enabled then the transceiver is busy. */
00053     return (TWCR & (1<<TWIE));
00054 }
00055
00056 /**
00057  * @brief Call this function to fetch the state information of the previous operation. The function will
00058  * hold execution (loop)
00059  * until the TWI_ISR has completed with the previous operation.
00060  * @return The TWI state code if there was an error.
00061  */
00062 void TWI_Get_State_Info(void)
00063 {
00064     /* Wait until TWI has completed the transmission. */
00065     while(TWI_Transceiver_Busy());
00066     /* Return the TWI state code if there was an error. */
00067     return TWI_state;
00068 }
00069 /**
00070  * @brief Call this function to send a prepared message. The first byte must contain the slave address and
00071  * the
00072  * read/write bit. Consecutive bytes contain the data to be sent, or empty locations for data to be read
00073  * from the slave. Also include how many bytes that should be sent/read including the address byte.
00074  * The function will hold execution (loop) until the TWI_ISR has completed with the previous operation,
00075  * then initialize the next operation and return.
00076  * @param msg Message buffer to be sent over the I2C bus.
00077  * @param msgSize Size of the message buffer to be sent over the I2C bus.
00078  */
00079 void TWI_Start_Transceiver_With_Data(uint8_t *msg, uint8_t msgSize)
00080 {
00081     /* Wait until TWI is ready for next transmission. */
00082     while(TWI_Transceiver_Busy());
00083
00084     /* Number of data to transmit. */
00085     TWI_msgSize = msgSize;
00086     /* Store slave address with R/W setting. */
00087     TWI_buf[0] = msg[0];
00088     /* If it is a write operation, then also copy data. */
00089     if(!(msg[0] & (TRUE<<TWI_READ_BIT)))
00090     {
00091         /* Loop through the indexes of the message buffer. */
00092         for(uint8_t i = 1; i<msgSize; i++)
00093         {
00094             /* Copy index i of the msg buffer to index i of TWI_buf*/
00095             TWI_buf[i] = msg[i];
00096         }
00097     }
00098
00099     /* Set the status register to zero*/
00100     TWI_statusReg = 0;
00101     /* Reset the TWI state. */
00102     TWI_state = TWI_NO_STATE;

```

```

00102     /* Set the TWCR to enable the TWI interface, enable TWI interrupt and clear the flag, and initiate the
00103     start condition. */
00103     TWCR = (1<<TWEN) | (1<<TWIE) | (1<<TWINT) | (0<<TWEA) | (1<<TWSTA) | (0<<TWSTO) | (0<<TWWC);
00104 }
00105
00106 /**
00107  * @brief Call this function to resend the last message. The driver will reuse the data previously put in
00108  * the transceiver buffers.
00109  * The function will hold execution (loop) until the TWI_ISR has completed with the previous operation,
00110  * then initialize the next operation and return.
00110  */
00111 void TWI_Start_Transceiver(void)
00112 {
00113     /* Wait until TWI is ready for next transmission. */
00114     while(TWI_Transceiver_Busy());
00115     /* Set the status register to zero*/
00116     TWI_statusReg = 0;
00117     /* Reset the TWI state. */
00118     TWI_state = TWI_NO_STATE;
00119     /* Set the TWCR to enable the TWI interface, enable TWI interrupt and clear the flag, and initiate the
00120     start condition. */
00120     TWCR = (1<<TWEN) | (1<<TWIE) | (1<<TWINT) | (0<<TWEA) | (1<<TWSTA) | (0<<TWSTO) | (0<<TWWC);
00121 }
00122
00123 /**
00124  * @brief Call this function to read out the requested data from the TWI transceiver buffer. I.e. first
00125  * call
00126  * TWI_Start_Transceiver to send a request for data to the slave. Then Run this function to collect the
00127  * data when they have arrived. Include a pointer to where to place the data and the number of bytes
00128  * requested (including the address field) in the function call. The function will hold execution (loop)
00129  * until the TWI_ISR has completed with the previous operation, before reading out the data and returning.
00130  * @param msg Message buffer to be received from the I2C bus.
00131  * @param msgSize Size of the message buffer to be received from the I2C bus.
00132  * @return
00132  */
00133 uint8_t TWI_Get_Data_From_Transceiver(uint8_t *msg, uint8_t msgSize)
00134 {
00135     /* Wait until TWI is ready for next transmission. */
00136     while(TWI_Transceiver_Busy());
00137     /* Last transmission competed successfully. */
00138     if(lastTransOK)
00139     {
00140         /* Loop through the indexes of the message buffer. */
00141         for(uint8_t i = 0; i<msgSize; i++)
00142         {
00143             /* Copy index i of the TWI_buf buffer to index i of msg. */
00144             msg[i] = TWI_buf[i];
00145         }
00146     }
00147     /* Return TWI error code if there was an error in the previous transmission the function. */
00148     return lastTransOK;
00149 }
00150
00151 /**
00152  * @brief This function is the Interrupt Service Routine (ISR), and called when the TWI interrupt is
00153  * triggered;
00154  * that is whenever a TWI event has occurred. This function should not be called directly from the main
00155  * application.
00155  */
00156 ISR(TWI_vect)
00157 {
00158     /* Pointer index for the TWI buffer. */
00159     static uint8_t TWI_bufPtr;
00160     switch(TWSR)
00161     {
00162         /* Start has been transmitted. */
00163         case TWI_START:
00164             /* Repeated START has been transmitted. */
00165             case TWI_REP_START:
00166                 /* Set buffer pointer to the TWI Address location. */
00167                 TWI_bufPtr = 0;
00168                 /* SLA+W has been transmitted and ACK received. */
00169                 case TWI_MTX_ADR_ACK:
00170                     /* Data byte has been transmitted and ACK received. */
00171                     case TWI_MTX_DATA_ACK:
00172                         /* Check if the pointer is less than last bit. */
00173                         if(TWI_bufPtr < TWI_msgSize)
00174                         {
00175                             /* Update the TWDR with the transceiver buffer information and increment the pointer index
00176                             */
00176                             TWDR = TWI_buf[TWI_bufPtr++];
00177                             /* Set the TWCR to enable the TWI interface and enable TWI interrupt and clear the flag to
00178                             send byte. */
00178                             TWCR = (1<<TWEN) | (1<<TWIE) | (1<<TWINT) | (0<<TWEA) | (0<<TWSTA) | (0<<TWSTO) | (0<<TWWC);
00179                         }
00180                         else
00181                         {

```

```

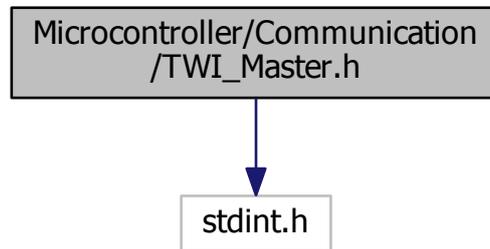
00182         /* Set status bits to completed successfully. */
00183         lastTransOK = TRUE;
00184         /* Set the TWCR to enable the TWI interface, disable TWI interrupt and clear the flag to
send byte, and initiate a stop condition. */
00185         TWCR = (1<<TWEN) | (0<<TWIE) | (1<<TWINT) | (0<<TWEA) | (0<<TWSTA) | (1<<TWSTO) | (0<<TWWC);
00186     }
00187     break;
00188     /* Data byte has been received and acknowledge transmitted. */
00189     case TWI_MR_X_DATA_ACK:
00190         /* Update the transceiver buffer with the TWDR information. */
00191         TWI_buf[TWI_bufPtr++] = TWDR;
00192         /* SLA+R has been transmitted and acknowledge received. */
00193     case TWI_MR_X_ADR_ACK:
00194         /* Check if the pointer is less than the second to last bit. */
00195         if(TWI_bufPtr < (TWI_msgSize-1))
00196         {
00197             /* Set the TWCR to enable the TWI interface, enable TWI Interrupt and clear the flag to
read next byte, and send acknowledge after reception. */
00198             TWCR = (1<<TWEN) | (1<<TWIE) | (1<<TWINT) | (1<<TWEA) | (0<<TWSTA) | (0<<TWSTO) | (0<<TWWC);
00199         }
00200         else
00201         {
00202             /* Set the TWCR to enable the TWI interface, enable TWI Interrupt and clear the flag to
read next byte, and send negative-acknowledge after reception. */
00203             TWCR = (1<<TWEN) | (1<<TWIE) | (1<<TWINT) | (0<<TWEA) | (0<<TWSTA) | (0<<TWSTO) | (0<<TWWC);
00204         }
00205         break;
00206     /* Data byte has been received and negative-acknowledge transmitted. */
00207     case TWI_MR_X_DATA_NACK:
00208         /* Update the transceiver buffer with the TWDR information. */
00209         TWI_buf[TWI_bufPtr] = TWDR;
00210         /* Set status bits to completed successfully. */
00211         lastTransOK = TRUE;
00212         /* Set the TWCR to enable the TWI interface, disable TWI Interrupt and clear the flag, and
initiate a stop condition. */
00213         TWCR = (1<<TWEN) | (0<<TWIE) | (1<<TWINT) | (0<<TWEA) | (0<<TWSTA) | (1<<TWSTO) | (0<<TWWC);
00214         break;
00215     /* Arbitration lost on the I2C bus. */
00216     case TWI_ARB_LOST:
00217         /* Set the TWCR to enable the TWI interface, enable TWI Interrupt and clear the flag, and
initiate a restart condition. */
00218         TWCR = (1<<TWEN) | (1<<TWIE) | (1<<TWINT) | (0<<TWEA) | (1<<TWSTA) | (0<<TWSTO) | (0<<TWWC);
00219         break;
00220     /* Bus error due to an illegal start or stop condition. */
00221     case TWI_BUS_ERROR:
00222         /* SLA+W has been transmitted and negative-acknowledge received. */
00223     case TWI_MTX_ADR_NACK:
00224         /* SLA+R has been transmitted and negative-acknowledge received. */
00225     case TWI_MR_X_ADR_NACK:
00226         /* Data byte has been transmitted and negative-acknowledge received. */
00227     case TWI_MTX_DATA_NACK:
00228     default:
00229         /* Store TWSR and automatically clears no Errors bit. */
00230         TWI_state = TWSR;
00231         /* Set the TWCR to enable the TWI interface and release TWI pins, disable TWI Interrupt flag,
and initiate a stop condition. */
00232         TWCR = (1<<TWEN) | (0<<TWIE) | (1<<TWINT) | (0<<TWEA) | (0<<TWSTA) | (1<<TWSTO) | (0<<TWWC);
00233     }
00234 }

```

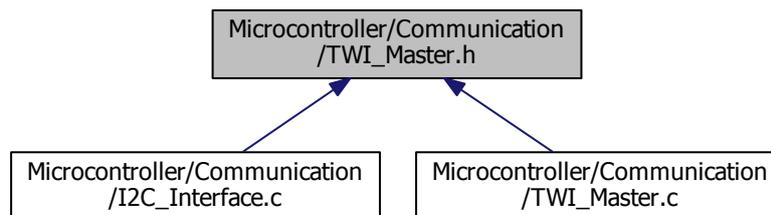
3.36 Microcontroller/Communication/TWI_Master.h File Reference

Header file for [TWI_master.c](#). Include this file in the application.

```
#include <stdint.h>
Include dependency graph for TWI_Master.h:
```



This graph shows which files directly or indirectly include this file:



Macros

- `#define TWI_BUFFER_SIZE 16`
Set this to the largest message size that will be sent including address byte.
- `#define TWI_BITRATE 400`
TWI_BITRATE in kHz.
- `#define F_CPU 16000000UL`
The current clock speed for the microcontroller.
- `#define TWI_PSBR (F_CPU/2000/TWI_BITRATE-8)`
TWI Bit rate Register setting.
- `#define TWI_TWPS 0x00`
- `#define TWI_TWBR TWI_PSBR`
- `#define MESSAGE 'e'`
- `#define TWI_READ_BIT 0`
Bit position for R/W bit in "address byte".
- `#define TWI_ADR_BITS 1`
Bit position for LSB of the slave address bits in the initial byte.
- `#define TRUE 1`
Boolean equivalent for TRUE.

- #define FALSE 0
Boolean equivalent for FALSE.
- #define TWI_START 0x08
START has been transmitted.
- #define TWI_REP_START 0x10
Repeated START has been transmitted.
- #define TWI_ARB_LOST 0x38
Arbitration lost.
- #define TWI_MTX_ADR_ACK 0x18
SLA+W has been transmitted and ACK received.
- #define TWI_MTX_ADR_NACK 0x20
SLA+W has been transmitted and NACK received.
- #define TWI_MTX_DATA_ACK 0x28
Data byte has been transmitted and ACK received.
- #define TWI_MTX_DATA_NACK 0x30
Data byte has been transmitted and NACK received.
- #define TWI_MRX_ADR_ACK 0x40
SLA+R has been transmitted and ACK received.
- #define TWI_MRX_ADR_NACK 0x48
SLA+R has been transmitted and NACK received.
- #define TWI_MRX_DATA_ACK 0x50
Data byte has been received and ACK transmitted.
- #define TWI_MRX_DATA_NACK 0x58
Data byte has been received and NACK transmitted.
- #define TWI_STX_ADR_ACK 0xA8
Own SLA+R has been received and ACK has been returned.
- #define TWI_STX_ADR_ACK_M_ARB_LOST 0xB0
Arbitration lost in SLA+R/W as Master, own SLA+R has been received, and ACK has been returned.
- #define TWI_STX_DATA_ACK 0xB8
Data byte in TWDR has been transmitted and ACK has been received.
- #define TWI_STX_DATA_NACK 0xC0
Data byte in TWDR has been transmitted and NOT ACK has been received.
- #define TWI_STX_DATA_ACK_LAST_BYTE 0xC8
Last data byte in TWDR has been transmitted (TWEA = '0') ACK has been received.
- #define TWI_SRX_ADR_ACK 0x60
Own SLA+W has been received ACK has been returned.
- #define TWI_SRX_ADR_ACK_M_ARB_LOST 0x68
Arbitration lost in SLA+R/W as Master, own SLA+W has been received, and ACK has been returned.
- #define TWI_SRX_GEN_ACK 0x70
General call address has been received and ACK has been returned.
- #define TWI_SRX_GEN_ACK_M_ARB_LOST 0x78
Arbitration lost in SLA+R/W as Master, General call address has been received, and ACK has been returned.
- #define TWI_SRX_ADR_DATA_ACK 0x80
Previously addressed with own SLA+W, data has been received, and ACK has been returned.
- #define TWI_SRX_ADR_DATA_NACK 0x88
Previously addressed with own SLA+W, data has been received, and NOT ACK has been returned.
- #define TWI_SRX_GEN_DATA_ACK 0x90
Previously addressed with general call, data has been received, and ACK has been returned.
- #define TWI_SRX_GEN_DATA_NACK 0x98
Previously addressed with general call, data has been received, and NOT ACK has been returned.
- #define TWI_SRX_STOP_RESTART 0xA0

A STOP condition or repeated START condition has been received while still addressed as Slave.

- #define [TWI_NO_STATE](#) 0xF8

No relevant state information available.

- #define [TWI_BUS_ERROR](#) 0x00

Bus error due to an illegal START or STOP condition.

Functions

- void [TWI_Master_Initialise](#) (void)

Call this function to set up the TWI master to its initial standby state.

- uint8_t [TWI_Transceiver_Busy](#) (void)

Call this function to test if the TWI_ISR is busy transmitting.

- uint8_t [TWI_Get_State_Info](#) (void)

Call this function to fetch the state information of the previous operation. The function will hold execution (loop) until the TWI_ISR has completed with the previous operation.

- void [TWI_Start_Transceiver_With_Data](#) (uint8_t *msg, uint8_t msgSize)

Call this function to send a prepared message. The first byte must contain the slave address and the read/write bit. Consecutive bytes contain the data to be sent, or empty locations for data to be read from the slave. Also include how many bytes that should be sent/read including the address byte. The function will hold execution (loop) until the TWI_ISR has completed with the previous operation, then initialize the next operation and return.

- void [TWI_Start_Transceiver](#) (void)

Call this function to resend the last message. The driver will reuse the data previously put in the transceiver buffers. The function will hold execution (loop) until the TWI_ISR has completed with the previous operation, then initialize the next operation and return.

- uint8_t [TWI_Get_Data_From_Transceiver](#) (uint8_t *msg, uint8_t msgSize)

Call this function to read out the requested data from the TWI transceiver buffer. I.e. first call [TWI_Start_Transceiver](#) to send a request for data to the slave. Then Run this function to collect the data when they have arrived. Include a pointer to where to place the data and the number of bytes requested (including the address field) in the function call. The function will hold execution (loop) until the TWI_ISR has completed with the previous operation, before reading out the data and returning.

3.36.1 Detailed Description

Header file for [TWI_master.c](#). Include this file in the application.

Atmel Corporation

Author

ltwa

Version

Revision: 1.13

Date

24. mai 2004 11:31:22

Definition in file [TWI_Master.h](#).

3.36.2 Macro Definition Documentation

3.36.2.1 #define F_CPU 16000000UL

The current clock speed for the microcontroller.

Definition at line 23 of file [TWI_Master.h](#).

3.36.2.2 #define FALSE 0

Boolean equivalent for FALSE.

Definition at line 59 of file [TWI_Master.h](#).

Referenced by [I2C_Interface_Single_Read\(\)](#), and [I2C_Interface_Write\(\)](#).

3.36.2.3 #define MESSAGE 'e'

Definition at line 48 of file [TWI_Master.h](#).

3.36.2.4 #define TRUE 1

Boolean equivalent for TRUE.

Definition at line 57 of file [TWI_Master.h](#).

Referenced by [I2C_Interface_Single_Read\(\)](#), [ISR\(\)](#), and [TWI_Start_Transceiver_With_Data\(\)](#).

3.36.2.5 #define TWI_ADR_BITS 1

Bit position for LSB of the slave address bits in the initial byte.

Definition at line 54 of file [TWI_Master.h](#).

3.36.2.6 #define TWI_ARB_LOST 0x38

Arbitration lost.

Definition at line 67 of file [TWI_Master.h](#).

Referenced by [ISR\(\)](#).

3.36.2.7 #define TWI_BITRATE 400

TWI_BITRATE in kHz.

Definition at line 19 of file [TWI_Master.h](#).

3.36.2.8 #define TWI_BUFFER_SIZE 16

Set this to the largest message size that will be sent including address byte.

Definition at line 17 of file [TWI_Master.h](#).

3.36.2.9 #define TWI_BUS_ERROR 0x00

Bus error due to an illegal START or STOP condition.

Definition at line 125 of file [TWI_Master.h](#).

Referenced by [ISR\(\)](#).

3.36.2.10 #define TWI_MRX_ADR_ACK 0x40

SLA+R has been transmitted and ACK received.

Definition at line 81 of file [TWI_Master.h](#).

Referenced by [ISR\(\)](#).

3.36.2.11 #define TWI_MR_X_ADR_NACK 0x48

SLA+R has been transmitted and NACK received.

Definition at line 83 of file [TWI_Master.h](#).

Referenced by [ISR\(\)](#).

3.36.2.12 #define TWI_MR_X_DATA_ACK 0x50

Data byte has been received and ACK transmitted.

Definition at line 85 of file [TWI_Master.h](#).

Referenced by [ISR\(\)](#).

3.36.2.13 #define TWI_MR_X_DATA_NACK 0x58

Data byte has been received and NACK transmitted.

Definition at line 87 of file [TWI_Master.h](#).

Referenced by [ISR\(\)](#).

3.36.2.14 #define TWI_MTX_ADR_ACK 0x18

SLA+W has been transmitted and ACK received.

Definition at line 71 of file [TWI_Master.h](#).

Referenced by [ISR\(\)](#).

3.36.2.15 #define TWI_MTX_ADR_NACK 0x20

SLA+W has been transmitted and NACK received.

Definition at line 73 of file [TWI_Master.h](#).

Referenced by [ISR\(\)](#).

3.36.2.16 #define TWI_MTX_DATA_ACK 0x28

Data byte has been transmitted and ACK received.

Definition at line 75 of file [TWI_Master.h](#).

Referenced by [ISR\(\)](#).

3.36.2.17 #define TWI_MTX_DATA_NACK 0x30

Data byte has been transmitted and NACK received.

Definition at line 77 of file [TWI_Master.h](#).

Referenced by [ISR\(\)](#).

3.36.2.18 #define TWI_NO_STATE 0xF8

No relevant state information available.

Definition at line 123 of file [TWI_Master.h](#).

Referenced by [TWI_Start_Transceiver\(\)](#), and [TWI_Start_Transceiver_With_Data\(\)](#).

3.36.2.19 #define TWI_PSBR (F_CPU/2000/TWI_BITRATE-8)

TWI Bit rate Register setting.

Definition at line 27 of file [TWI_Master.h](#).

3.36.2.20 #define TWI_READ_BIT 0

Bit position for R/W bit in "address byte".

Definition at line 52 of file [TWI_Master.h](#).

Referenced by [I2C_Interface_Single_Read\(\)](#), [I2C_Interface_Write\(\)](#), and [TWI_Start_Transceiver_With_Data\(\)](#).

3.36.2.21 #define TWI_REP_START 0x10

Repeated START has been transmitted.

Definition at line 65 of file [TWI_Master.h](#).

Referenced by [ISR\(\)](#).

3.36.2.22 #define TWI_SRX_ADR_ACK 0x60

Own SLA+W has been received ACK has been returned.

Definition at line 103 of file [TWI_Master.h](#).

3.36.2.23 #define TWI_SRX_ADR_ACK_M_ARB_LOST 0x68

Arbitration lost in SLA+R/W as Master, own SLA+W has been received, and ACK has been returned.

Definition at line 105 of file [TWI_Master.h](#).

3.36.2.24 #define TWI_SRX_ADR_DATA_ACK 0x80

Previously addressed with own SLA+W, data has been received, and ACK has been returned.

Definition at line 111 of file [TWI_Master.h](#).

3.36.2.25 #define TWI_SRX_ADR_DATA_NACK 0x88

Previously addressed with own SLA+W, data has been received, and NOT ACK has been returned.

Definition at line 113 of file [TWI_Master.h](#).

3.36.2.26 #define TWI_SRX_GEN_ACK 0x70

General call address has been received and ACK has been returned.

Definition at line 107 of file [TWI_Master.h](#).

3.36.2.27 #define TWI_SRX_GEN_ACK_M_ARB_LOST 0x78

Arbitration lost in SLA+R/W as Master, General call address has been received, and ACK has been returned.

Definition at line 109 of file [TWI_Master.h](#).

3.36.2.28 #define TWI_SRX_GEN_DATA_ACK 0x90

Previously addressed with general call, data has been received, and ACK has been returned.

Definition at line 115 of file [TWI_Master.h](#).

3.36.2.29 #define TWI_SRX_GEN_DATA_NACK 0x98

Previously addressed with general call, data has been received, and NOT ACK has been returned.

Definition at line 117 of file [TWI_Master.h](#).

3.36.2.30 #define TWI_SRX_STOP_RESTART 0xA0

A STOP condition or repeated START condition has been received while still addressed as Slave.

Definition at line 119 of file [TWI_Master.h](#).

3.36.2.31 #define TWI_START 0x08

START has been transmitted.

Definition at line 63 of file [TWI_Master.h](#).

Referenced by [ISR\(\)](#).

3.36.2.32 #define TWI_STX_ADR_ACK 0xA8

Own SLA+R has been received and ACK has been returned.

Definition at line 91 of file [TWI_Master.h](#).

3.36.2.33 #define TWI_STX_ADR_ACK_M_ARB_LOST 0xB0

Arbitration lost in SLA+R/W as Master, own SLA+R has been received, and ACK has been returned.

Definition at line 93 of file [TWI_Master.h](#).

3.36.2.34 #define TWI_STX_DATA_ACK 0xB8

Data byte in TWDR has been transmitted and ACK has been received.

Definition at line 95 of file [TWI_Master.h](#).

3.36.2.35 #define TWI_STX_DATA_ACK_LAST_BYTE 0xC8

Last data byte in TWDR has been transmitted (TWEA = '0') ACK has been received.

Definition at line 99 of file [TWI_Master.h](#).

3.36.2.36 #define TWI_STX_DATA_NACK 0xC0

Data byte in TWDR has been transmitted and NOT ACK has been received.

Definition at line 97 of file [TWI_Master.h](#).

3.36.2.37 #define TWI_TWBR TWI_PSBR

Definition at line 47 of file [TWI_Master.h](#).

Referenced by [TWI_Master_Initialise\(\)](#).

3.36.2.38 #define TWI_TWPS 0x00

Definition at line 46 of file [TWI_Master.h](#).

Referenced by [TWI_Master_Initialise\(\)](#).

3.36.3 Function Documentation

3.36.3.1 uint8_t TWI_Get_Data_From_Transceiver (uint8_t * msg, uint8_t msgSize)

Call this function to read out the requested data from the TWI transceiver buffer. I.e. first call [TWI_Start_Transceiver](#) to send a request for data to the slave. Then Run this function to collect the data when they have arrived. Include a pointer to where to place the data and the number of bytes requested (including the address field) in the function call. The function will hold execution (loop) until the [TWI_ISR](#) has completed with the previous operation, before reading out the data and returning.

Parameters

<i>msg</i>	Message buffer to be received from the I2C bus.
<i>msgSize</i>	Size of the message buffer to be received from the I2C bus.

Returns

Definition at line 133 of file [TWI_Master.c](#).

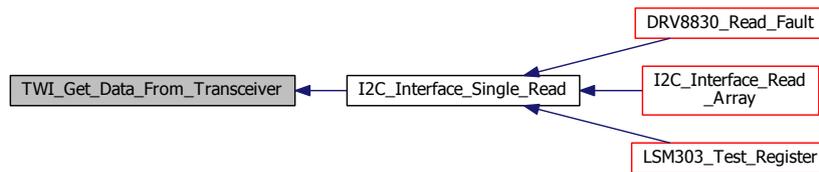
References [TWI_Transceiver_Busy\(\)](#).

Referenced by [I2C_Interface_Single_Read\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.36.3.2 `uint8_t TWI_Get_State_Info (void)`

Call this function to fetch the state information of the previous operation. The function will hold execution (loop) until the `TWI_ISR` has completed with the previous operation.

Returns

The TWI state code if there was an error.

Definition at line 61 of file [TWI_Master.c](#).

References [TWI_Transceiver_Busy\(\)](#).

Here is the call graph for this function:



3.36.3.3 `void TWI_Master_Initialise (void)`

Call this function to set up the TWI master to its initial standby state.

Warning

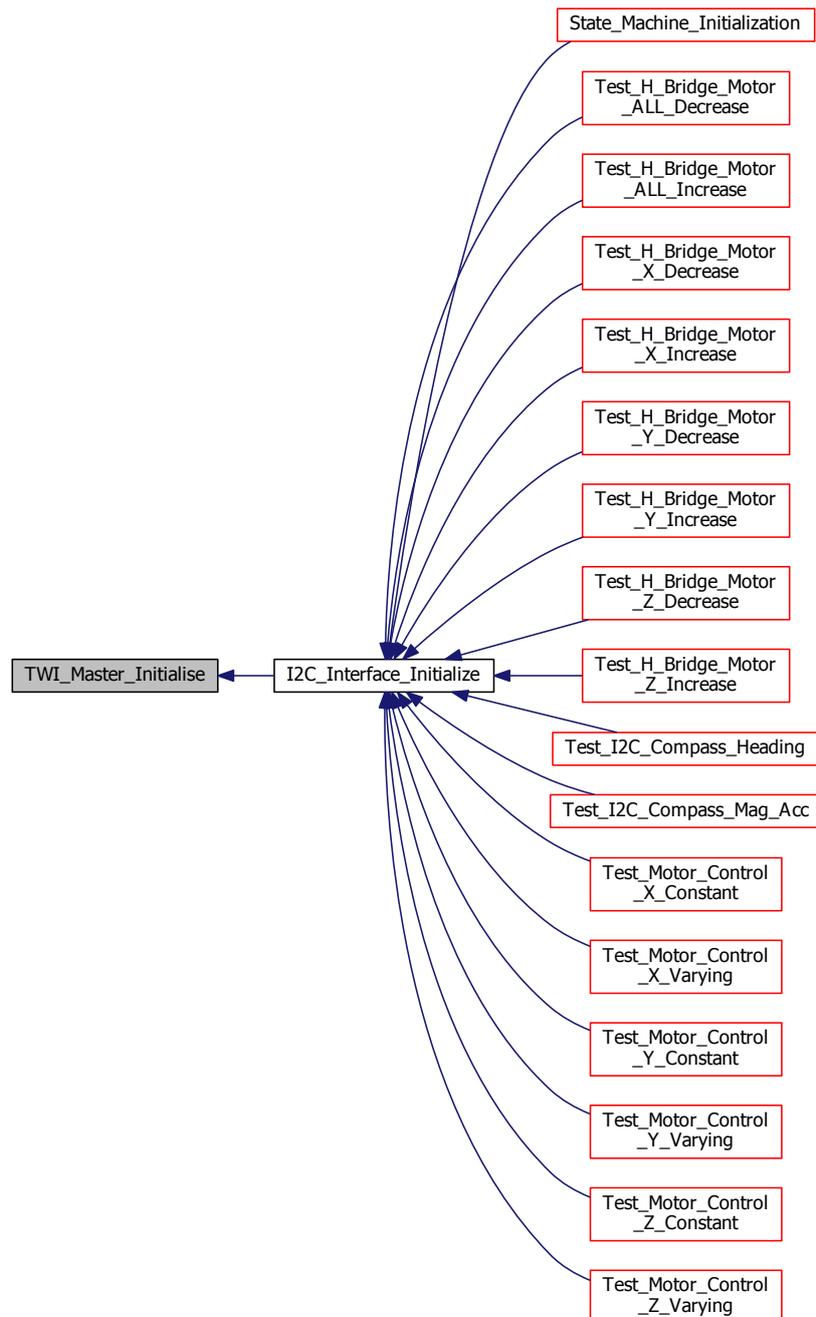
Remember to enable interrupts from the main application after initializing the TWI.

Definition at line 34 of file [TWI_Master.c](#).

References [TWI_TWBR](#), and [TWI_TWPS](#).

Referenced by [I2C_Interface_Initialize\(\)](#).

Here is the caller graph for this function:



3.36.3.4 void TWI_Start_Transceiver (void)

Call this function to resend the last message. The driver will reuse the data previously put in the transceiver buffers. The function will hold execution (loop) until the TWI_ISR has completed with the previous operation, then initialize the next operation and return.

Definition at line 111 of file [TWI_Master.c](#).

References [TWI_NO_STATE](#), and [TWI_Transceiver_Busy\(\)](#).

Here is the call graph for this function:



3.36.3.5 void TWI_Start_Transceiver_With_Data (uint8_t * msg, uint8_t msgSize)

Call this function to send a prepared message. The first byte must contain the slave address and the read/write bit. Consecutive bytes contain the data to be sent, or empty locations for data to be read from the slave. Also include how many bytes that should be sent/read including the address byte. The function will hold execution (loop) until the TWI_ISR has completed with the previous operation, then initialize the next operation and return.

Parameters

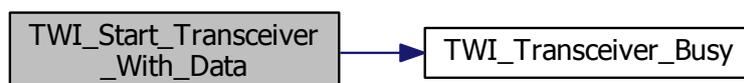
<i>msg</i>	Message buffer to be sent over the I2C bus.
<i>msgSize</i>	Size of the message buffer to be sent over the I2C bus.

Definition at line 78 of file [TWI_Master.c](#).

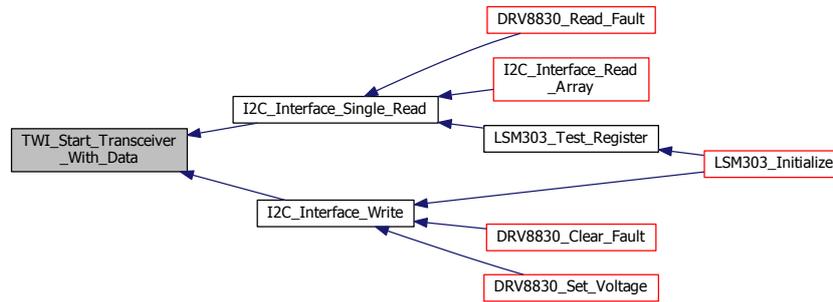
References [TRUE](#), [TWI_NO_STATE](#), [TWI_READ_BIT](#), and [TWI_Transceiver_Busy\(\)](#).

Referenced by [I2C_Interface_Single_Read\(\)](#), and [I2C_Interface_Write\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.36.3.6 uint8_t TWI_Transceiver_Busy (void)

Call this function to test if the TWI_ISR is busy transmitting.

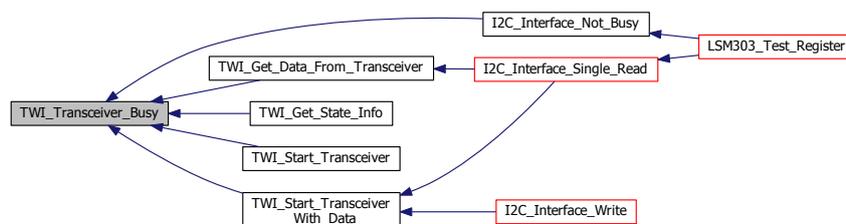
Returns

Status of the transceiver.

Definition at line 50 of file [TWI_Master.c](#).

Referenced by [I2C_Interface_Not_Busy\(\)](#), [TWI_Get_Data_From_Transceiver\(\)](#), [TWI_Get_State_Info\(\)](#), [TWI_Start_Transceiver\(\)](#), and [TWI_Start_Transceiver_With_Data\(\)](#).

Here is the caller graph for this function:



3.37 TWI_Master.h

```

00001 /**
00002  * Atmel Corporation
00003  * @file TWI_master.h
00004  * @author ltwa
00005  * @version Revision: 1.13
00006  * @date 24. mai 2004 11:31:22
00007  * @brief Header file for TWI_master.c.
00008  * Include this file in the application.
00009  */
00010
00011 #ifndef TWI_MASTER_H_
00012 #define TWI_MASTER_H_
00013
00014 #include <stdint.h>
00015
00016 /** @brief Set this to the largest message size that will be sent including address byte. */
  
```

```
00017 #define TWI_BUFFER_SIZE 16
00018 /** @brief TWI_BITRATE in kHz. */
00019 #define TWI_BITRATE 400
00020
00021 #ifndef F_CPU
00022     /** @brief The current clock speed for the microcontroller. */
00023     #define F_CPU 16000000UL
00024 #endif
00025
00026 /** @brief TWI Bit rate Register setting. */
00027 #define TWI_PSBR (F_CPU/2000/TWI_BITRATE-8)
00028
00029 #if TWI_PSBR > 0x3FC0
00030     #define TWI_TWPS 0x03
00031     #define TWI_TWBR 0xFF
00032     #define MESSAGE 'a'
00033 #elif TWI_PSBR > 0x0FF0
00034     #define TWI_TWPS 0x03
00035     #define TWI_TWBR TWI_PSBR/64
00036     #define MESSAGE 'b'
00037 #elif TWI_PSBR > 0x03FC
00038     #define TWI_TWPS 0x02
00039     #define TWI_TWBR TWI_PSBR/16
00040     #define MESSAGE 'c'
00041 #elif TWI_PSBR > 0x00FF
00042     #define TWI_TWPS 0x01
00043     #define TWI_TWBR TWI_PSBR/4
00044     #define MESSAGE 'd'
00045 #else
00046     #define TWI_TWPS 0x00
00047     #define TWI_TWBR TWI_PSBR
00048     #define MESSAGE 'e'
00049 #endif
00050
00051 /** @brief Bit position for R/W bit in "address byte". */
00052 #define TWI_READ_BIT 0
00053 /** @brief Bit position for LSB of the slave address bits in the initial byte. */
00054 #define TWI_ADR_BITS 1
00055
00056 /** @brief Boolean equivalent for TRUE. */
00057 #define TRUE 1
00058 /** @brief Boolean equivalent for FALSE. */
00059 #define FALSE 0
00060
00061 /* General TWI Master status codes. */
00062 /** @brief START has been transmitted. */
00063 #define TWI_START 0x08
00064 /** @brief Repeated START has been transmitted. */
00065 #define TWI_REP_START 0x10
00066 /** @brief Arbitration lost. */
00067 #define TWI_ARB_LOST 0x38
00068
00069 /* TWI Master Transmitter status codes. */
00070 /** @brief SLA+W has been transmitted and ACK received. */
00071 #define TWI_MTX_ADR_ACK 0x18
00072 /** @brief SLA+W has been transmitted and NACK received. */
00073 #define TWI_MTX_ADR_NACK 0x20
00074 /** @brief Data byte has been transmitted and ACK received. */
00075 #define TWI_MTX_DATA_ACK 0x28
00076 /** @brief Data byte has been transmitted and NACK received. */
00077 #define TWI_MTX_DATA_NACK 0x30
00078
00079 /* TWI Master Receiver status codes. */
00080 /** @brief SLA+R has been transmitted and ACK received. */
00081 #define TWI_MRX_ADR_ACK 0x40
00082 /** @brief SLA+R has been transmitted and NACK received. */
00083 #define TWI_MRX_ADR_NACK 0x48
00084 /** @brief Data byte has been received and ACK transmitted. */
00085 #define TWI_MRX_DATA_ACK 0x50
00086 /** @brief Data byte has been received and NACK transmitted. */
00087 #define TWI_MRX_DATA_NACK 0x58
00088
00089 /* TWI Slave Transmitter status codes. */
00090 /** @brief Own SLA+R has been received and ACK has been returned. */
00091 #define TWI_STX_ADR_ACK 0xA8
00092 /** @brief Arbitration lost in SLA+R/W as Master, own SLA+R has been received, and ACK has been returned.
00093 */
00093 #define TWI_STX_ADR_ACK_M_ARB_LOST 0xB0
00094 /** @brief Data byte in TWDR has been transmitted and ACK has been received. */
00095 #define TWI_STX_DATA_ACK 0xB8
00096 /** @brief Data byte in TWDR has been transmitted and NOT ACK has been received. */
00097 #define TWI_STX_DATA_NACK 0xC0
00098 /** @brief Last data byte in TWDR has been transmitted (TWEA = '0') ACK has been received. */
00099 #define TWI_STX_DATA_ACK_LAST_BYTE 0xC8
00100
00101 /* TWI Slave Receiver status codes. */
00102 /** @brief Own SLA+W has been received ACK has been returned. */
```

```

00103 #define TWI_SRX_ADR_ACK 0x60
00104 /** @brief Arbitration lost in SLA+R/W as Master, own SLA+W has been received, and ACK has been returned.
*/
00105 #define TWI_SRX_ADR_ACK_M_ARB_LOST 0x68
00106 /** @brief General call address has been received and ACK has been returned. */
00107 #define TWI_SRX_GEN_ACK 0x70
00108 /** @brief Arbitration lost in SLA+R/W as Master, General call address has been received, and ACK has been
returned. */
00109 #define TWI_SRX_GEN_ACK_M_ARB_LOST 0x78
00110 /** @brief Previously addressed with own SLA+W, data has been received, and ACK has been returned. */
00111 #define TWI_SRX_ADR_DATA_ACK 0x80
00112 /** @brief Previously addressed with own SLA+W, data has been received, and NOT ACK has been returned. */
00113 #define TWI_SRX_ADR_DATA_NACK 0x88
00114 /** @brief Previously addressed with general call, data has been received, and ACK has been returned. */
00115 #define TWI_SRX_GEN_DATA_ACK 0x90
00116 /** @brief Previously addressed with general call, data has been received, and NOT ACK has been returned.
*/
00117 #define TWI_SRX_GEN_DATA_NACK 0x98
00118 /** @brief A STOP condition or repeated START condition has been received while still addressed as Slave.
*/
00119 #define TWI_SRX_STOP_RESTART 0xA0
00120
00121 /* TWI Miscellaneous status codes. */
00122 /** @brief No relevant state information available. */
00123 #define TWI_NO_STATE 0xF8
00124 /** @brief Bus error due to an illegal START or STOP condition. */
00125 #define TWI_BUS_ERROR 0x00
00126
00127 void TWI_Master_Initialise(void);
00128 uint8_t TWI_Transceiver_Busy(void);
00129 uint8_t TWI_Get_State_Info(void);
00130 void TWI_Start_Transceiver_With_Data(uint8_t *msg, uint8_t msgSize);
00131 void TWI_Start_Transceiver(void);
00132 uint8_t TWI_Get_Data_From_Transceiver(uint8_t *msg, uint8_t msgSize);
00133
00134 #endif

```

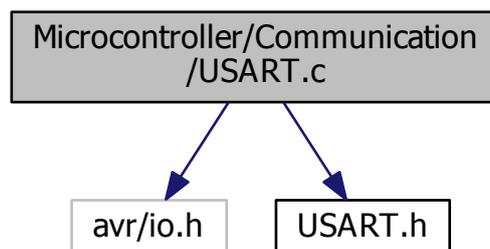
3.38 Microcontroller/Communication/USART.c File Reference

This file is code to control the USART on the ATmega328P.

```
#include <avr/io.h>
```

```
#include "USART.h"
```

Include dependency graph for USART.c:



Macros

- #define **FOSC** 16000000
The current clock speed for the microcontroller.
- #define **BAUD** 76800
The desired baud rate for the USART connection.

- `#define MYUBRR (FOSC/(16UL*BAUD)-1)`
The desired baud rate for the USART connection.
- `#define PRECISION 3`
The desired precision for the conversion between float and string.

Functions

- void `USART_Initialize` (void)
Initializes USART on the ATmega328P.
- void `USART_Send_Byte` (char data)
Sends a single byte using USART.
- unsigned char `USART_Receive_Byte` (void)
Receives a single byte using USART.
- void `USART_Send_String` (char *string)
Sends a c style string using USART.
- int `USART_Read_String` (char *string)
Receives a c style string using USART.
- int `USART_Read_String_With_Echo` (char *string)
Receives a c style string using USART, then send the c style string.
- char * `USART_ftoa` (float Input_Value, char *Buffer)
Converts a given floating point value to a string.

3.38.1 Detailed Description

This file is code to control the USART on the ATmega328P.

Author

Ryan Lipski, Nick Sikkema

Version

Revision: 2.0

Date

Last Updated: 4/21/2015
Created: 11/12/2014 7:36:35 PM

Definition in file [USART.c](#).

3.38.2 Macro Definition Documentation

3.38.2.1 `#define BAUD 76800`

The desired baud rate for the USART connection.

Definition at line [12](#) of file [USART.c](#).

3.38.2.2 `#define FOSC 16000000`

The current clock speed for the microcontroller.

Definition at line [10](#) of file [USART.c](#).

3.38.2.3 #define MYUBRR (FOSC/(16UL*BAUD)-1)

The desired baud rate for the USART connection.

Definition at line 14 of file [USART.c](#).

Referenced by [USART_Initialize\(\)](#).

3.38.2.4 #define PRECISION 3

The desired precision for the conversion between float and string.

Definition at line 16 of file [USART.c](#).

Referenced by [USART_ftoa\(\)](#).

3.38.3 Function Documentation

3.38.3.1 char* USART_ftoa (float *Input_Value*, char * *Buffer*)

Converts a given floating point value to a string.

Parameters

<i>Input_Value</i>	The floating point value to be converted.
<i>Buffer</i>	The buffer pointer for the character array.

Returns

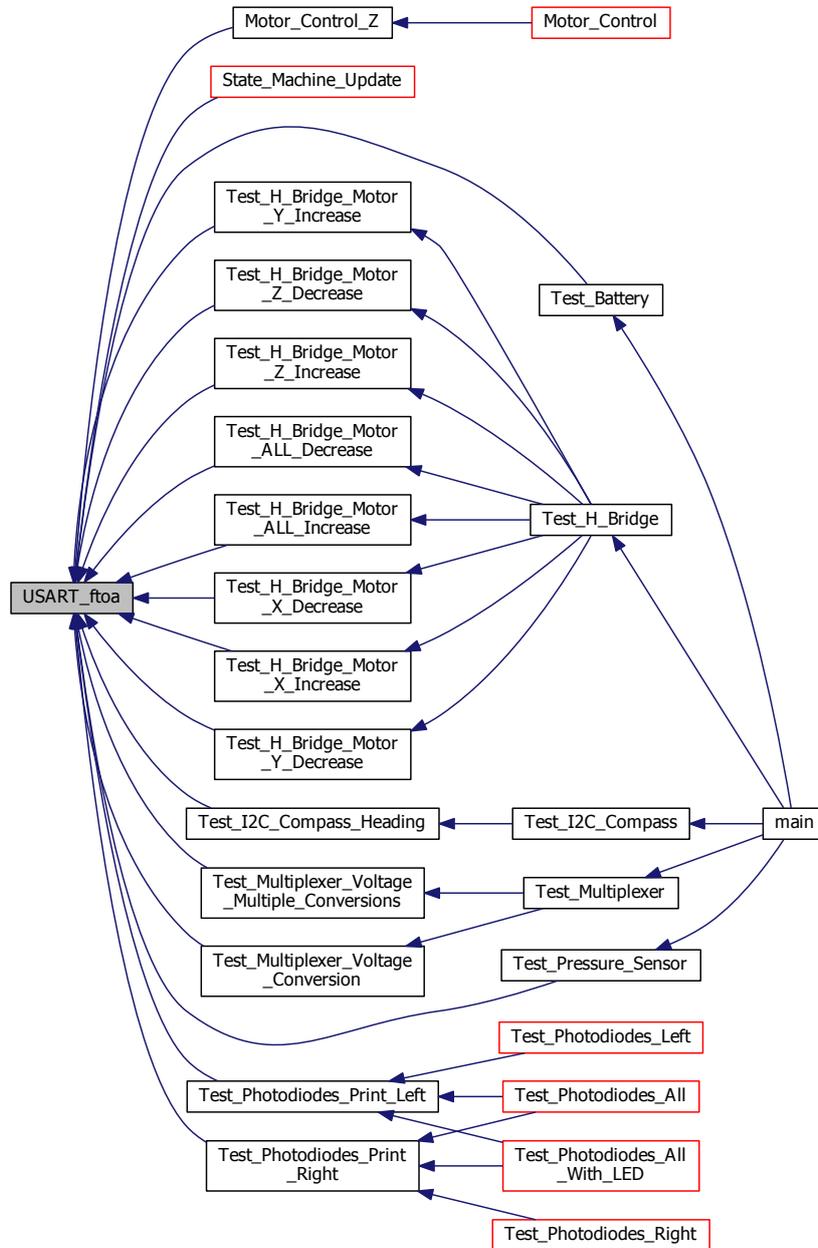
The floating point value in string form.

Definition at line 136 of file [USART.c](#).

References [FLOATING_POINT_BUFFER_SIZE](#), and [PRECISION](#).

Referenced by [Motor_Control_Z\(\)](#), [State_Machine_Update\(\)](#), [Test_Battery\(\)](#), [Test_H_Bridge_Motor_ALL_↔Decrease\(\)](#), [Test_H_Bridge_Motor_ALL_Increase\(\)](#), [Test_H_Bridge_Motor_X_Decrease\(\)](#), [Test_H_Bridge_↔Motor_X_Increase\(\)](#), [Test_H_Bridge_Motor_Y_Decrease\(\)](#), [Test_H_Bridge_Motor_Y_Increase\(\)](#), [Test_H_Bridge_↔Motor_Z_Decrease\(\)](#), [Test_H_Bridge_Motor_Z_Increase\(\)](#), [Test_I2C_Compass_Heading\(\)](#), [Test_Multiplexer_↔Voltage_Conversion\(\)](#), [Test_Multiplexer_Voltage_Multiple_Conversions\(\)](#), [Test_Photodiodes_Print_Left\(\)](#), [Test_↔Photodiodes_Print_Right\(\)](#), and [Test_Pressure_Sensor\(\)](#).

Here is the caller graph for this function:



3.38.3.2 void USART_Initialize (void)

Initializes USART on the ATmega328P.

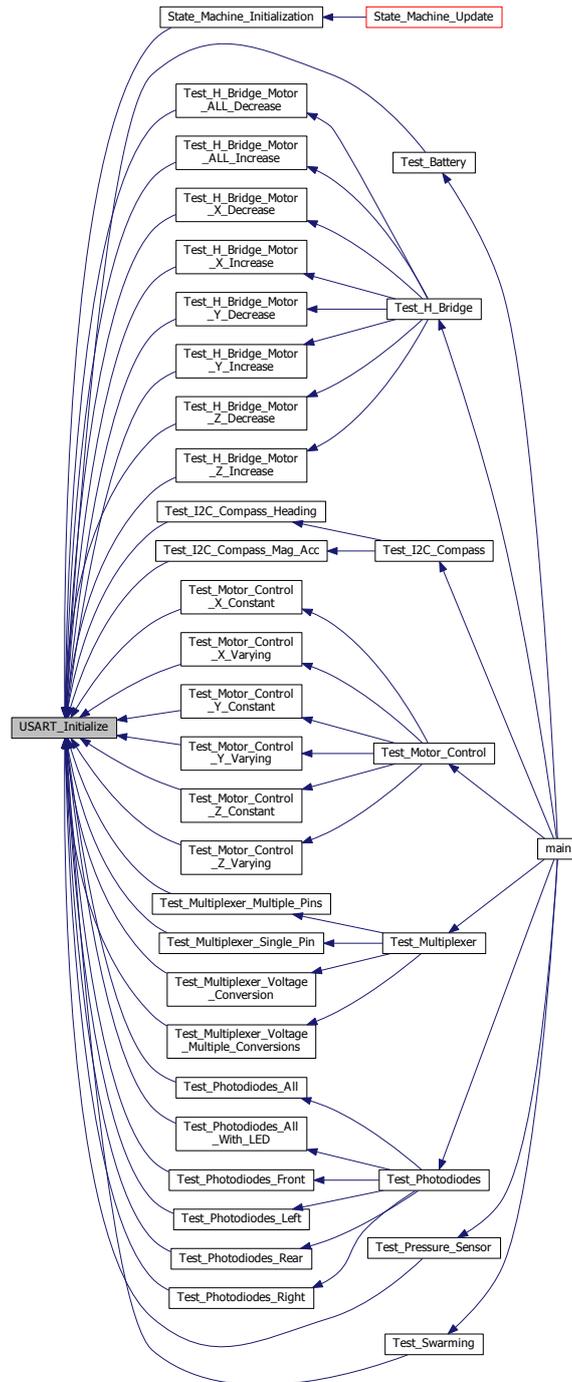
Definition at line 24 of file [USART.c](#).

References [MYUBRR](#).

Referenced by [State_Machine_Initialization\(\)](#), [Test_Battery\(\)](#), [Test_H_Bridge_Motor_ALL_Decrease\(\)](#), [Test_H_Bridge_Motor_ALL_Increase\(\)](#), [Test_H_Bridge_Motor_X_Decrease\(\)](#), [Test_H_Bridge_Motor_X_Increase\(\)](#), [Test_H_Bridge_Motor_Y_Decrease\(\)](#), [Test_H_Bridge_Motor_Y_Increase\(\)](#), [Test_H_Bridge_Motor_Z_Decrease\(\)](#), [Test_](#)

`_H_Bridge_Motor_Z_Increase()`, `Test_I2C_Compass_Heading()`, `Test_I2C_Compass_Mag_Acc()`, `Test_Motor_↔Control_X_Constant()`, `Test_Motor_Control_X_Varying()`, `Test_Motor_Control_Y_Constant()`, `Test_Motor_Control_↔_Y_Varying()`, `Test_Motor_Control_Z_Constant()`, `Test_Motor_Control_Z_Varying()`, `Test_Multiplexer_Multiple_↔Pins()`, `Test_Multiplexer_Single_Pin()`, `Test_Multiplexer_Voltage_Conversion()`, `Test_Multiplexer_Voltage_Multiple_↔Conversions()`, `Test_Photodiodes_All()`, `Test_Photodiodes_All_With_LED()`, `Test_Photodiodes_Front()`, `Test_↔Photodiodes_Left()`, `Test_Photodiodes_Rear()`, `Test_Photodiodes_Right()`, `Test_Pressure_Sensor()`, and `Test_↔Swarming()`.

Here is the caller graph for this function:



3.38.3.3 int USART_Read_String (char * *string*)

Receives a c style string using USART.

Parameters

<i>string</i>	The c style string that is to be received.
---------------	--

Returns

The number of characters that are received.

Receive the information from the keyboard and save it in a temporary variable.

Definition at line 79 of file [USART.c](#).

References [USART_Receive_Byte\(\)](#).

Here is the call graph for this function:



3.38.3.4 int USART_Read_String_With_Echo (char * *string*)

Receives a c style string using USART, then send the c style string.

Parameters

<i>string</i>	The c style string that is to be received.
---------------	--

Returns

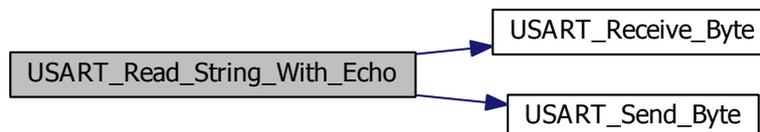
The number of characters that are received.

Receive the information from the keyboard and save it in a temporary variable.

Definition at line 106 of file [USART.c](#).

References [USART_Receive_Byte\(\)](#), and [USART_Send_Byte\(\)](#).

Here is the call graph for this function:



3.38.3.5 unsigned char USART_Receive_Byte (void)

Receives a single byte using USART.

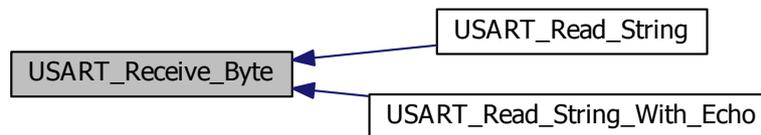
Returns

Character that is received.

Definition at line 53 of file [USART.c](#).

Referenced by [USART_Read_String\(\)](#), and [USART_Read_String_With_Echo\(\)](#).

Here is the caller graph for this function:

**3.38.3.6 void USART_Send_Byte (char data)**

Sends a single byte using USART.

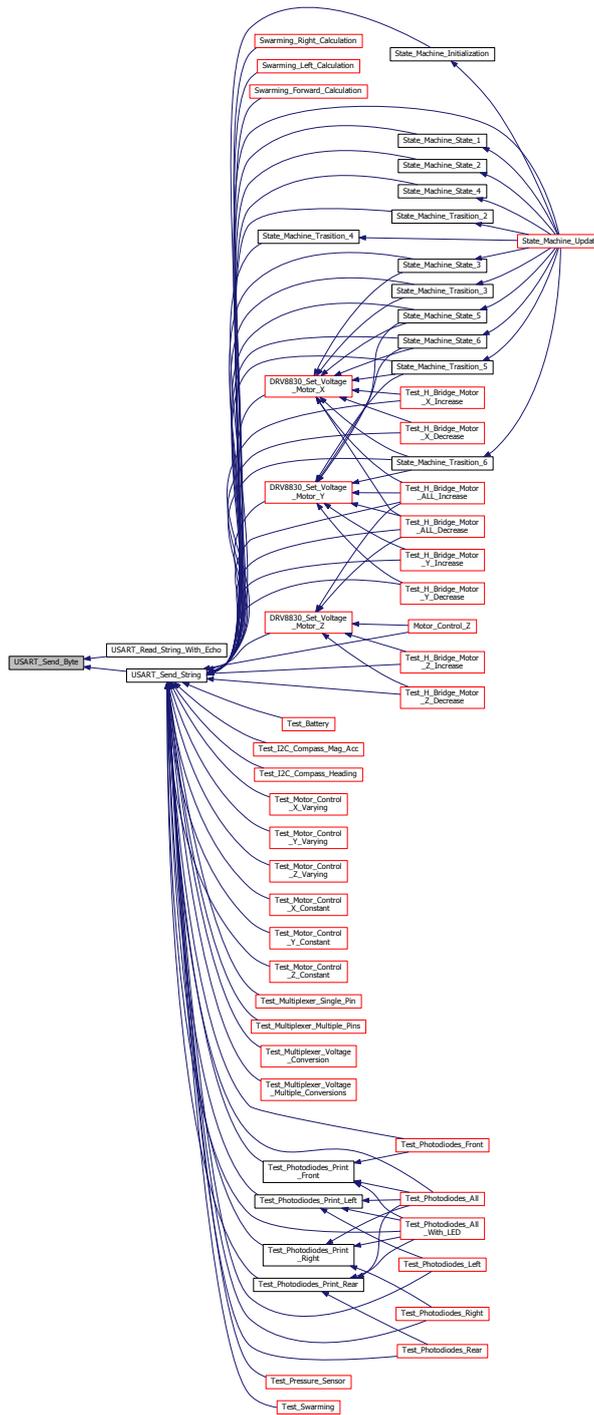
Parameters

<i>data</i>	Character that is to be sent.
-------------	-------------------------------

Definition at line 41 of file [USART.c](#).

Referenced by [USART_Read_String_With_Echo\(\)](#), and [USART_Send_String\(\)](#).

Here is the caller graph for this function:



3.38.3.7 void USART_Send_String (char * string)

Sends a c style string using USART.

Parameters

<i>string</i>	The c style string that is to be sent.
---------------	--

Definition at line 65 of file [USART.c](#).

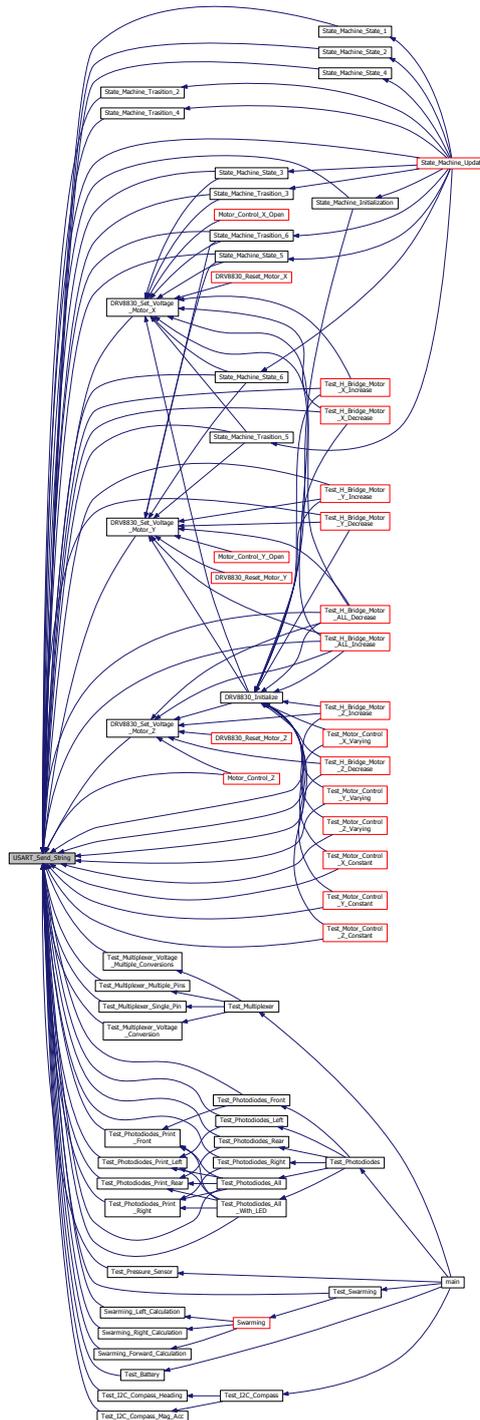
References [USART_Send_Byte\(\)](#).

Referenced by [DRV8830_Set_Voltage_Motor_X\(\)](#), [DRV8830_Set_Voltage_Motor_Y\(\)](#), [DRV8830_Set_Voltage_Motor_Z\(\)](#), [Motor_Control_Z\(\)](#), [State_Machine_Initialization\(\)](#), [State_Machine_State_1\(\)](#), [State_Machine_State_2\(\)](#), [State_Machine_State_3\(\)](#), [State_Machine_State_4\(\)](#), [State_Machine_State_5\(\)](#), [State_Machine_State_6\(\)](#), [State_Machine_Trasition_2\(\)](#), [State_Machine_Trasition_3\(\)](#), [State_Machine_Trasition_4\(\)](#), [State_Machine_Trasition_5\(\)](#), [State_Machine_Trasition_6\(\)](#), [State_Machine_Update\(\)](#), [Swarming_Forward_Calculation\(\)](#), [Swarming_Left_Calculation\(\)](#), [Swarming_Right_Calculation\(\)](#), [Test_Battery\(\)](#), [Test_H_Bridge_Motor_ALL_Decrease\(\)](#), [Test_H_Bridge_Motor_ALL_Increase\(\)](#), [Test_H_Bridge_Motor_X_Decrease\(\)](#), [Test_H_Bridge_Motor_X_Increase\(\)](#), [Test_H_Bridge_Motor_Y_Decrease\(\)](#), [Test_H_Bridge_Motor_Y_Increase\(\)](#), [Test_H_Bridge_Motor_Z_Decrease\(\)](#), [Test_H_Bridge_Motor_Z_Increase\(\)](#), [Test_I2C_Compass_Heading\(\)](#), [Test_I2C_Compass_Mag_Acc\(\)](#), [Test_Motor_Control_X_Constant\(\)](#), [Test_Motor_Control_X_Varying\(\)](#), [Test_Motor_Control_Y_Constant\(\)](#), [Test_Motor_Control_Y_Varying\(\)](#), [Test_Motor_Control_Z_Constant\(\)](#), [Test_Motor_Control_Z_Varying\(\)](#), [Test_Multiplexer_Multiple_Pins\(\)](#), [Test_Multiplexer_Single_Pin\(\)](#), [Test_Multiplexer_Voltage_Conversion\(\)](#), [Test_Multiplexer_Voltage_Multiple_Conversions\(\)](#), [Test_Photodiodes_All\(\)](#), [Test_Photodiodes_All_With_LED\(\)](#), [Test_Photodiodes_Front\(\)](#), [Test_Photodiodes_Left\(\)](#), [Test_Photodiodes_Print_Front\(\)](#), [Test_Photodiodes_Print_Left\(\)](#), [Test_Photodiodes_Print_Rear\(\)](#), [Test_Photodiodes_Print_Right\(\)](#), [Test_Photodiodes_Rear\(\)](#), [Test_Photodiodes_Right\(\)](#), [Test_Pressure_Sensor\(\)](#), and [Test_Swarming\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.39 USART.c

```

00001 /**
00002  * @file USART.c
00003  * @author Ryan Lipski, Nick Sikkema
00004  * @version Revision: 2.0
00005  * @date Last Updated: 4/21/2015\n Created: 11/12/2014 7:36:35 PM
00006  * @brief This file is code to control the USART on the ATmega328P.
00007  */
    
```

```

00008
00009 /** @brief The current clock speed for the microcontroller. */
00010 #define FOSC 16000000
00011 /** @brief The desired baud rate for the USART connection. */
00012 #define BAUD 76800
00013 /** @brief The desired baud rate for the USART connection. */
00014 #define MYUBRR (FOSC/(16UL*BAUD)-1)
00015 /** @brief The desired precision for the conversion between float and string. */
00016 #define PRECISION 3
00017
00018 #include <avr/io.h>
00019 #include "USART.h"
00020
00021 /**
00022  * @brief Initializes USART on the ATmega328P.
00023  */
00024 void USART_Initialize(void)
00025 {
00026     /* Set baud rate. */
00027     UBRR0H = (unsigned char)(MYUBRR>>8);
00028     UBRR0L = (unsigned char)MYUBRR;
00029     /* Clear UCSR0A. This is to make sure that the USART is in single asynchronous mode. */
00030     UCSR0A = 0x00;
00031     /* Enable receiver and transmitter. */
00032     UCSROB = (1<<RXEN0)|(1<<TXEN0);
00033     /* Set frame format: 8 data and 1 stop bit. */
00034     UCSROC = (3<<UCSZ00);
00035 }
00036
00037 /**
00038  * @brief Sends a single byte using USART.
00039  * @param data Character that is to be sent.
00040  */
00041 void USART_Send_Byte(char data)
00042 {
00043     /* Wait for empty transmit buffer. */
00044     while(!(UCSR0A & (1<<UDRE0)));
00045     /* Put data into buffer, sends the data. */
00046     UDR0 = data;
00047 }
00048
00049 /**
00050  * @brief Receives a single byte using USART.
00051  * @return Character that is received.
00052  */
00053 unsigned char USART_Receive_Byte(void)
00054 {
00055     /* Wait for data to be received. */
00056     while(!(UCSR0A & (1<<RXC0)));
00057     /* Get and return received data from buffer. */
00058     return(UDR0);
00059 }
00060
00061 /**
00062  * @brief Sends a c style string using USART.
00063  * @param string The c style string that is to be sent.
00064  */
00065 void USART_Send_String(char* string)
00066 {
00067     /* Loop through the string pointer. */
00068     for(char* pointer=string; *pointer!=0x00; pointer++) {
00069         /* Send the character. */
00070         USART_Send_Byte(*pointer);
00071     }
00072 }
00073
00074 /**
00075  * @brief Receives a c style string using USART.
00076  * @param string The c style string that is to be received.
00077  * @return The number of characters that are received.
00078  */
00079 int USART_Read_String(char* string)
00080 {
00081     /* A variable to keep track of the string length. */
00082     int count=0;
00083     while (1) {
00084         /* Receive the information from the keyboard and save it in a temporary variable. */
00085         char collector=USART_Receive_Byte();
00086         /* Break if the enter key is pressed. Check to see if the carriage return and line feed is sent. */
00087         if (collector=='\n' || collector=='\r'){
00088             break;
00089         }
00090         /* Update the string with the byte received. */
00091         *string++=collector;
00092         /* Increase the length count. */
00093         count++;
00094     }

```

```

00095     /* terminate the char array string with 0x00/ */
00096     *string=0x00;
00097     /* Return the length of the string. */
00098     return(count);
00099 }
00100
00101 /**
00102  * @brief Receives a c style string using USART, then send the c style string.
00103  * @param string The c style string that is to be received.
00104  * @return The number of characters that are received.
00105  */
00106 int USART_Read_String_With_Echo(char* string)
00107 {
00108     /* A variable to keep track of the string length. */
00109     int count=0;
00110     while(1) {
00111         /** Receive the information from the keyboard and save it in a temporary variable. */
00112         char collector=USART_Receive_Byte();
00113         /* Send back the information just received to the screen. */
00114         USART_Send_Byte(collector);
00115         /* Break if the enter key is pressed. Check to see if the carriage return and line feed is sent. */
00116         if (collector=='\n' || collector=='\r'){
00117             break;
00118         }
00119         /* Update the string with the byte received. */
00120         *string++=collector;
00121         /* Increase the length count. */
00122         count++;
00123     }
00124     /* terminate the char array string with 0x00/ */
00125     *string=0x00;
00126     /* Return the length of the string. */
00127     return(count);
00128 }
00129
00130 /**
00131  * @brief Converts a given floating point value to a string.
00132  * @param Input_Value The floating point value to be converted.
00133  * @param Buffer The buffer pointer for the character array.
00134  * @return The floating point value in string form.
00135  */
00136 char* USART_ftoa(float Input_Value, char* Buffer)
00137 {
00138     /* Initialize a index variable. */
00139     unsigned int i = 0;
00140     /* Loop through the indexes and reinitialize the buffer. */
00141     for(i = 0; i<FLOATING_POINT_BUFFER_SIZE; i++)
00142     {
00143         /* Set the buffer index to NULL. */
00144         Buffer[i] = 0;
00145     }
00146     /* Copy the Input_Value to a local variable. Removing the decimal value. */
00147     int Temp_Value = Input_Value;
00148     /* Copy the Input_Value to a local variable. */
00149     float Temp_Input_Value = Input_Value;
00150     /* Reset the index pointer. */
00151     i = 0;
00152     /* Check to see if the Temp_Input_Value is negative. */
00153     if(Temp_Input_Value<0)
00154     {
00155         /* If the Input_Value is negative then flip sign. */
00156         Temp_Value = -Temp_Value;
00157         Temp_Input_Value = -Temp_Input_Value;
00158         /* Add a negative sign*/
00159         Buffer[i++] = '-';
00160     }
00161     /* Process individual digits */
00162     if (Temp_Value==0)
00163     {
00164         /* Add zero to the array. */
00165         Buffer[i++] = 48;
00166     }
00167     else
00168     {
00169         /* Copy the integer portion of the Input_Value to the Buffer. */
00170         while (Temp_Value != 0)
00171         {
00172             /* Add the current digit to the array. */
00173             Buffer[i++] = (char)(Temp_Value%10+48);
00174             /* Remove the last digit. */
00175             Temp_Value = Temp_Value/10;
00176         }
00177     }
00178     /* Add the decimal place. */
00179     Buffer[i++] = '.';
00180     /* Reset the Temp_Value. */
00181     Temp_Value = (int)Temp_Input_Value;

```

```

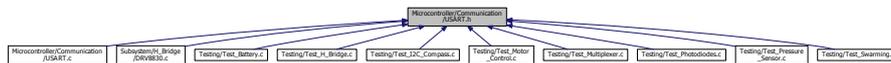
00182     /* Initialize a precision counter. */
00183     int Precision_Count = 0;
00184     do {
00185         /* Update the floating portion by subtracting the integer portion. */
00186         Temp_Input_Value -= Temp_Value;
00187         /* Update the integer portion by shifting the decimal place and remove the decimal places. */
00188         Temp_Value = Temp_Input_Value*10;
00189         /* Append the new integer value to the Buffer. */
00190         Buffer[i++] = (char)(Temp_Value+48);
00191         /* Update the precision count. */
00192         Precision_Count++;
00193         /* Update the floating portion by shifting the decimal place. */
00194         Temp_Input_Value *= 10;
00195         /* Check to see if the index is greater than the buffer size or that the precision is good enough.
*/
00196     } while(i < FLOATING_POINT_BUFFER_SIZE && Precision_Count < PRECISION);
00197     /* Return the buffer pointer. */
00198     return(Buffer);
00199 }

```

3.40 Microcontroller/Communication/USART.h File Reference

This is the header file to control the USART on the ATmega328P.

This graph shows which files directly or indirectly include this file:



Macros

- `#define FLOATING_POINT_BUFFER_SIZE 15`
Max buffer size for the conversion between float and string.

Functions

- void `USART_Initialize` (void)
Initializes USART on the ATmega328P.
- void `USART_Send_Byte` (char data)
Sends a single byte using USART.
- unsigned char `USART_Receive_Byte` (void)
Receives a single byte using USART.
- void `USART_Send_String` (char *string)
Sends a c style string using USART.
- int `USART_Read_String` (char *string)
Receives a c style string using USART.
- int `USART_Read_String_With_Echo` (char *string)
Receives a c style string using USART, then send the c style string.
- char * `USART_ftoa` (float Input_Value, char *Buffer)
Converts a given floating point value to a string.

3.40.1 Detailed Description

This is the header file to control the USART on the ATmega328P.

Author

Ryan Lipski, Nick Sikkema

Version

Revision: 1.5

Date

Last Updated: 3/20/2015

Created: 11/12/2014 7:36:35 PM

Definition in file [USART.h](#).

3.40.2 Macro Definition Documentation**3.40.2.1 #define FLOATING_POINT_BUFFER_SIZE 15**

Max buffer size for the conversion between float and string.

Definition at line 13 of file [USART.h](#).

Referenced by [Motor_Control_Z\(\)](#), [State_Machine_Update\(\)](#), [Test_Battery\(\)](#), [Test_H_Bridge_Motor_ALL_Decrease\(\)](#), [Test_H_Bridge_Motor_ALL_Increase\(\)](#), [Test_H_Bridge_Motor_X_Decrease\(\)](#), [Test_H_Bridge_Motor_X_Increase\(\)](#), [Test_H_Bridge_Motor_Y_Decrease\(\)](#), [Test_H_Bridge_Motor_Y_Increase\(\)](#), [Test_H_Bridge_Motor_Z_Decrease\(\)](#), [Test_H_Bridge_Motor_Z_Increase\(\)](#), [Test_I2C_Compass_Heading\(\)](#), [Test_Multiplexer_Voltage_Conversion\(\)](#), [Test_Multiplexer_Voltage_Multiple_Conversions\(\)](#), [Test_Photodiodes_Print_Left\(\)](#), [Test_Photodiodes_Print_Right\(\)](#), [Test_Pressure_Sensor\(\)](#), and [USART_ftoa\(\)](#).

3.40.3 Function Documentation**3.40.3.1 char* USART_ftoa (float *Input_Value*, char * *Buffer*)**

Converts a given floating point value to a string.

Parameters

<i>Input_Value</i>	The floating point value to be converted.
<i>Buffer</i>	The buffer pointer for the character array.

Returns

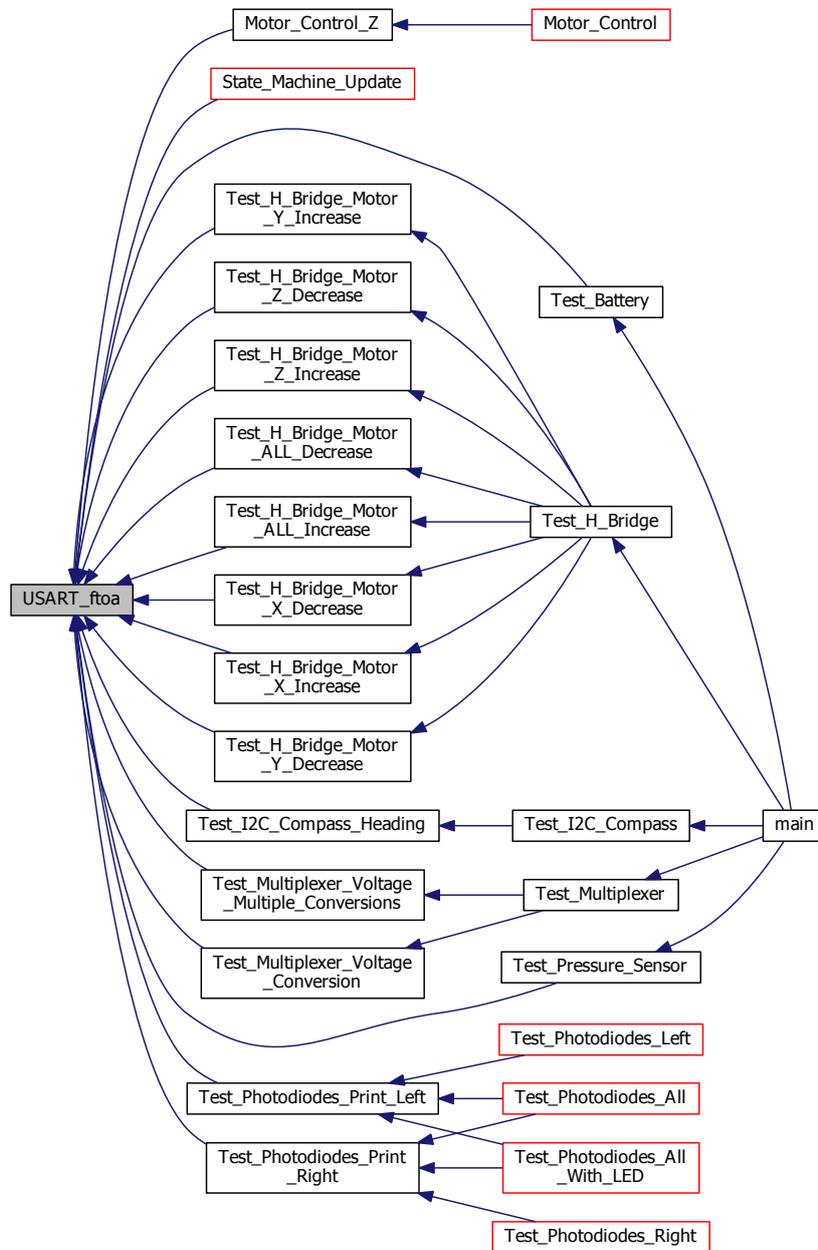
The floating point value in string form.

Definition at line 136 of file [USART.c](#).

References [FLOATING_POINT_BUFFER_SIZE](#), and [PRECISION](#).

Referenced by [Motor_Control_Z\(\)](#), [State_Machine_Update\(\)](#), [Test_Battery\(\)](#), [Test_H_Bridge_Motor_ALL_Decrease\(\)](#), [Test_H_Bridge_Motor_ALL_Increase\(\)](#), [Test_H_Bridge_Motor_X_Decrease\(\)](#), [Test_H_Bridge_Motor_X_Increase\(\)](#), [Test_H_Bridge_Motor_Y_Decrease\(\)](#), [Test_H_Bridge_Motor_Y_Increase\(\)](#), [Test_H_Bridge_Motor_Z_Decrease\(\)](#), [Test_H_Bridge_Motor_Z_Increase\(\)](#), [Test_I2C_Compass_Heading\(\)](#), [Test_Multiplexer_Voltage_Conversion\(\)](#), [Test_Multiplexer_Voltage_Multiple_Conversions\(\)](#), [Test_Photodiodes_Print_Left\(\)](#), [Test_Photodiodes_Print_Right\(\)](#), and [Test_Pressure_Sensor\(\)](#).

Here is the caller graph for this function:



3.40.3.2 void USART_Initialize (void)

Initializes USART on the ATmega328P.

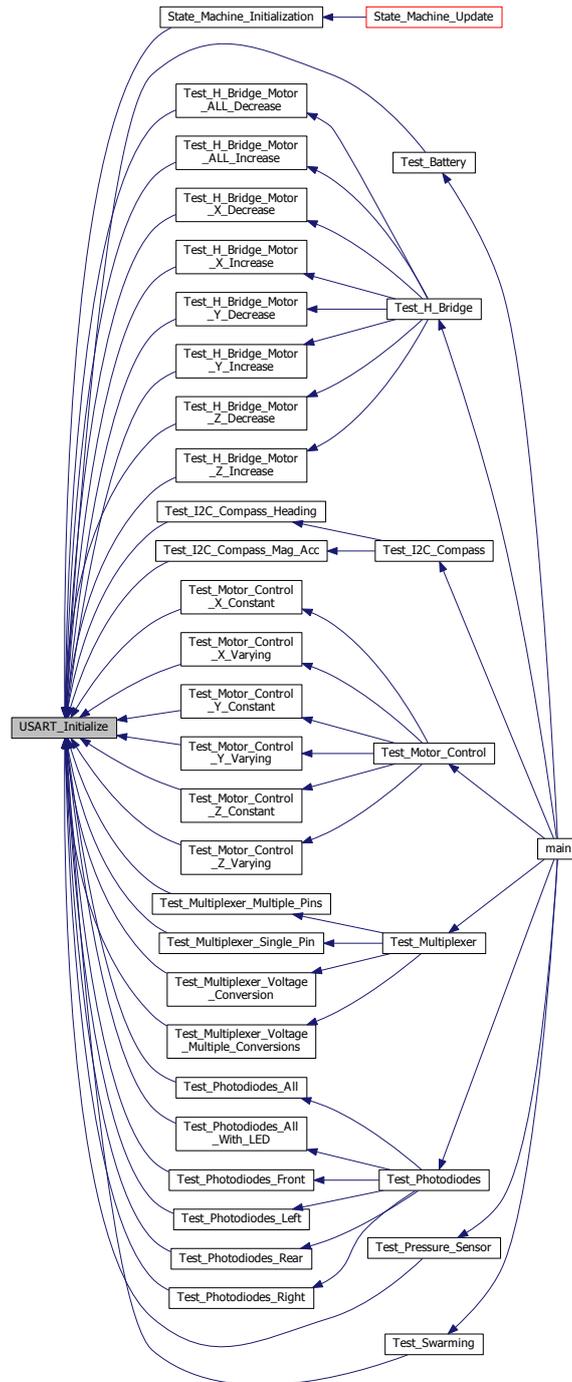
Definition at line 24 of file [USART.c](#).

References [MYUBRR](#).

Referenced by [State_Machine_Initialization\(\)](#), [Test_Battery\(\)](#), [Test_H_Bridge_Motor_ALL_Decrease\(\)](#), [Test_H_Bridge_Motor_ALL_Increase\(\)](#), [Test_H_Bridge_Motor_X_Decrease\(\)](#), [Test_H_Bridge_Motor_X_Increase\(\)](#), [Test_H_Bridge_Motor_Y_Decrease\(\)](#), [Test_H_Bridge_Motor_Y_Increase\(\)](#), [Test_H_Bridge_Motor_Z_Decrease\(\)](#), [Test_](#)

`_H_Bridge_Motor_Z_Increase()`, `Test_I2C_Compass_Heading()`, `Test_I2C_Compass_Mag_Acc()`, `Test_Motor_Control_X_Constant()`, `Test_Motor_Control_X_Varying()`, `Test_Motor_Control_Y_Constant()`, `Test_Motor_Control_Y_Varying()`, `Test_Motor_Control_Z_Constant()`, `Test_Motor_Control_Z_Varying()`, `Test_Multiplexer_Multiple_Pins()`, `Test_Multiplexer_Single_Pin()`, `Test_Multiplexer_Voltage_Conversion()`, `Test_Multiplexer_Voltage_Multiple_Conversions()`, `Test_Photodiodes_All()`, `Test_Photodiodes_All_With_LED()`, `Test_Photodiodes_Front()`, `Test_Photodiodes_Left()`, `Test_Photodiodes_Rear()`, `Test_Photodiodes_Right()`, `Test_Pressure_Sensor()`, and `Test_Swarming()`.

Here is the caller graph for this function:



3.40.3.3 int USART_Read_String (char * *string*)

Receives a c style string using USART.

Parameters

<i>string</i>	The c style string that is to be received.
---------------	--

Returns

The number of characters that are received.

Receive the information from the keyboard and save it in a temporary variable.

Definition at line 79 of file [USART.c](#).

References [USART_Receive_Byte\(\)](#).

Here is the call graph for this function:



3.40.3.4 int USART_Read_String_With_Echo (char * *string*)

Receives a c style string using USART, then send the c style string.

Parameters

<i>string</i>	The c style string that is to be received.
---------------	--

Returns

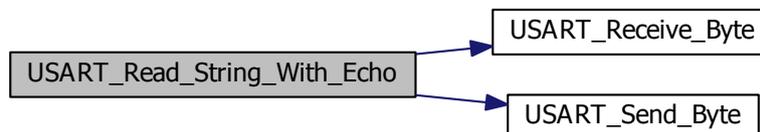
The number of characters that are received.

Receive the information from the keyboard and save it in a temporary variable.

Definition at line 106 of file [USART.c](#).

References [USART_Receive_Byte\(\)](#), and [USART_Send_Byte\(\)](#).

Here is the call graph for this function:



3.40.3.5 unsigned char USART_Receive_Byte (void)

Receives a single byte using USART.

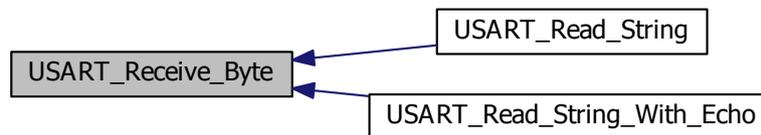
Returns

Character that is received.

Definition at line 53 of file [USART.c](#).

Referenced by [USART_Read_String\(\)](#), and [USART_Read_String_With_Echo\(\)](#).

Here is the caller graph for this function:

**3.40.3.6 void USART_Send_Byte (char data)**

Sends a single byte using USART.

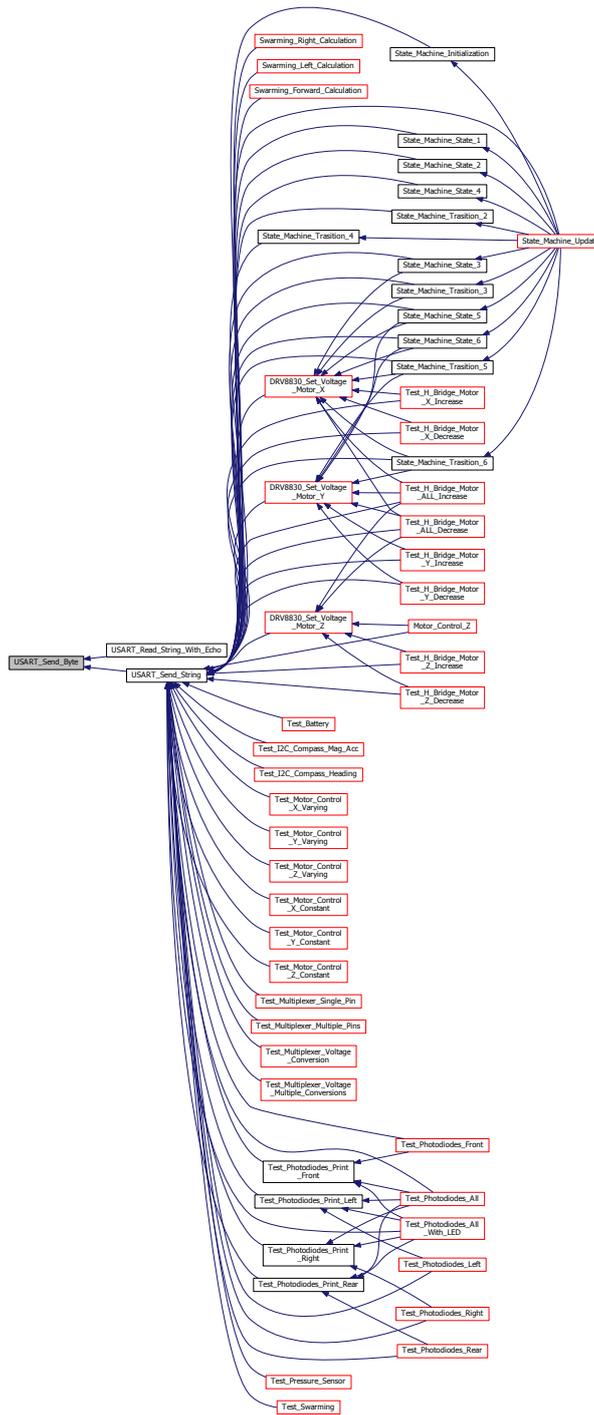
Parameters

<i>data</i>	Character that is to be sent.
-------------	-------------------------------

Definition at line 41 of file [USART.c](#).

Referenced by [USART_Read_String_With_Echo\(\)](#), and [USART_Send_String\(\)](#).

Here is the caller graph for this function:



3.40.3.7 void USART_Send_String (char * string)

Sends a c style string using USART.

Parameters

<i>string</i>	The c style string that is to be sent.
---------------	--

Definition at line 65 of file [USART.c](#).

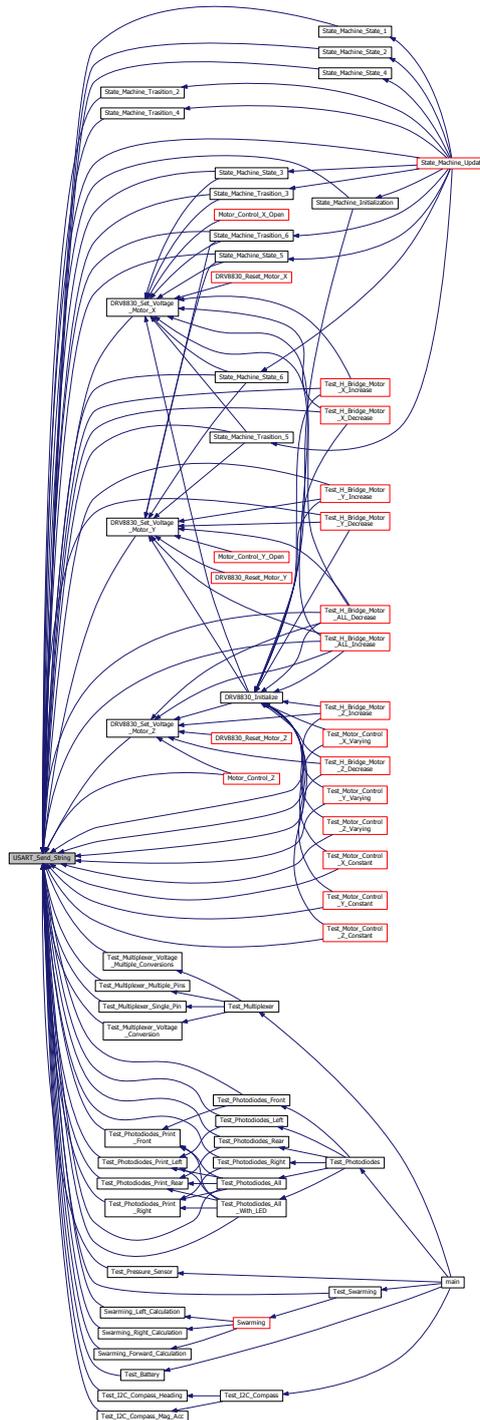
References [USART_Send_Byte\(\)](#).

Referenced by [DRV8830_Set_Voltage_Motor_X\(\)](#), [DRV8830_Set_Voltage_Motor_Y\(\)](#), [DRV8830_Set_Voltage_Motor_Z\(\)](#), [Motor_Control_Z\(\)](#), [State_Machine_Initialization\(\)](#), [State_Machine_State_1\(\)](#), [State_Machine_State_2\(\)](#), [State_Machine_State_3\(\)](#), [State_Machine_State_4\(\)](#), [State_Machine_State_5\(\)](#), [State_Machine_State_6\(\)](#), [State_Machine_Trasition_2\(\)](#), [State_Machine_Trasition_3\(\)](#), [State_Machine_Trasition_4\(\)](#), [State_Machine_Trasition_5\(\)](#), [State_Machine_Trasition_6\(\)](#), [State_Machine_Update\(\)](#), [Swarming_Forward_Calculation\(\)](#), [Swarming_Left_Calculation\(\)](#), [Swarming_Right_Calculation\(\)](#), [Test_Battery\(\)](#), [Test_H_Bridge_Motor_ALL_Decrease\(\)](#), [Test_H_Bridge_Motor_ALL_Increase\(\)](#), [Test_H_Bridge_Motor_X_Decrease\(\)](#), [Test_H_Bridge_Motor_X_Increase\(\)](#), [Test_H_Bridge_Motor_Y_Decrease\(\)](#), [Test_H_Bridge_Motor_Y_Increase\(\)](#), [Test_H_Bridge_Motor_Z_Decrease\(\)](#), [Test_H_Bridge_Motor_Z_Increase\(\)](#), [Test_I2C_Compass_Heading\(\)](#), [Test_I2C_Compass_Mag_Acc\(\)](#), [Test_Motor_Control_X_Constant\(\)](#), [Test_Motor_Control_X_Varying\(\)](#), [Test_Motor_Control_Y_Constant\(\)](#), [Test_Motor_Control_Y_Varying\(\)](#), [Test_Motor_Control_Z_Constant\(\)](#), [Test_Motor_Control_Z_Varying\(\)](#), [Test_Multiplexer_Multiple_Pins\(\)](#), [Test_Multiplexer_Single_Pin\(\)](#), [Test_Multiplexer_Voltage_Conversion\(\)](#), [Test_Multiplexer_Voltage_Multiple_Conversions\(\)](#), [Test_Photodiodes_All\(\)](#), [Test_Photodiodes_All_With_LED\(\)](#), [Test_Photodiodes_Front\(\)](#), [Test_Photodiodes_Left\(\)](#), [Test_Photodiodes_Print_Front\(\)](#), [Test_Photodiodes_Print_Left\(\)](#), [Test_Photodiodes_Print_Rear\(\)](#), [Test_Photodiodes_Print_Right\(\)](#), [Test_Photodiodes_Rear\(\)](#), [Test_Photodiodes_Right\(\)](#), [Test_Pressure_Sensor\(\)](#), and [Test_Swarming\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.41 USART.h

```

00001 /**
00002  * @file USART.h
00003  * @author Ryan Lipski, Nick Sikkema
00004  * @version Revision: 1.5
00005  * @date Last Updated: 3/20/2015\n Created: 11/12/2014 7:36:35 PM
00006  * @brief This is the header file to control the USART on the ATmega328P.
00007  */
    
```

```

00008
00009 #ifndef USART_H_
00010 #define USART_H_
00011
00012 /** @brief Max buffer size for the conversion between float and string. */
00013 #define FLOATING_POINT_BUFFER_SIZE 15
00014
00015 void USART_Initialize(void);
00016 void USART_Send_Byte(char data);
00017 unsigned char USART_Receive_Byte(void);
00018 void USART_Send_String(char* string);
00019 int USART_Read_String(char* string);
00020 int USART_Read_String_With_Echo(char* string);
00021 char * USART_ftoa(float Input_Value, char *Buffer);
00022
00023 #endif /* USART_H_ */

```

3.42 Microcontroller/Interrupt/Interrupt.c File Reference

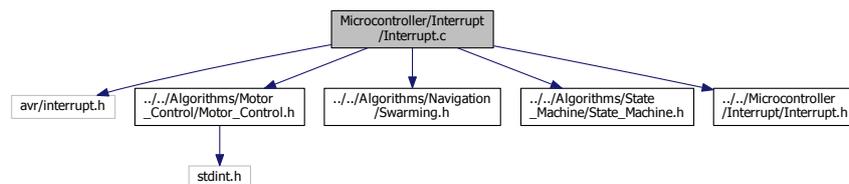
This file is code for the interrupts on the ATmega328P.

```

#include <avr/interrupt.h>
#include "../Algorithms/Motor_Control/Motor_Control.h"
#include "../Algorithms/Navigation/Swarming.h"
#include "../Algorithms/State_Machine/State_Machine.h"
#include "../Microcontroller/Interrupt/Interrupt.h"

```

Include dependency graph for Interrupt.c:



Functions

- [ISR \(TIMER0_COMPA_vect\)](#)
Interrupt for timer 0 compare A vector, which is used to update the motor control value.
- [ISR \(TIMER1_COMPA_vect\)](#)
Interrupt for timer 1 compare A vector, which is used to update the swarming value.
- [ISR \(TIMER2_COMPA_vect\)](#)
Interrupt for timer 2 compare A vector, which is used to update the state machine.

3.42.1 Detailed Description

This file is code for the interrupts on the ATmega328P.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 1/29/2015
Created: 1/27/2015 4:38:39 PM

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 1/29/2015
Created: 1/27/2015 4:28:45 PM

Definition in file [Interrupt.c](#).

3.42.2 Function Documentation

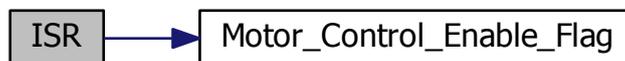
3.42.2.1 ISR (TIMER0_COMPA_vect)

Interrupt for timer 0 compare A vector, which is used to update the motor control value.

Definition at line 18 of file [Interrupt.c](#).

References [Motor_Control_Enable_Flag\(\)](#).

Here is the call graph for this function:



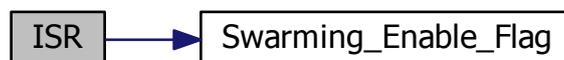
3.42.2.2 ISR (TIMER1_COMPA_vect)

Interrupt for timer 1 compare A vector, which is used to update the swarming value.

Definition at line 27 of file [Interrupt.c](#).

References [Swarming_Enable_Flag\(\)](#).

Here is the call graph for this function:



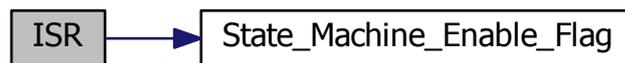
3.42.2.3 ISR (TIMER2_COMPA_vect)

Interrupt for timer 2 compare A vector, which is used to update the state machine.

Definition at line 36 of file [Interrupt.c](#).

References [State_Machine_Enable_Flag\(\)](#).

Here is the call graph for this function:



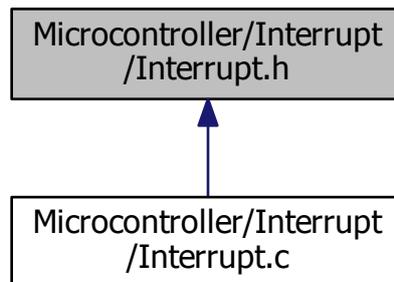
3.43 Interrupt.c

```

00001 /**
00002  * @file Interrupt.c
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 1/29/2015\n Created: 1/27/2015 4:38:39 PM
00006  * @brief This file is code for the interrupts on the ATmega328P.
00007  */
00008
00009 #include <avr/interrupt.h>
00010 #include "../Algorithms/Motor_Control/Motor_Control.h"
00011 #include "../Algorithms/Navigation/Swarming.h"
00012 #include "../Algorithms/State_Machine/State_Machine.h"
00013 #include "../Microcontroller/Interrupt/Interrupt.h"
00014
00015 /**
00016  * @brief Interrupt for timer 0 compare A vector, which is used to update the motor control value.
00017  */
00018 ISR(TIMER0_COMPA_vect)
00019 {
00020     /* Enable the flag to update the motor control values. */
00021     Motor_Control_Enable_Flag();
00022 }
00023
00024 /**
00025  * @brief Interrupt for timer 1 compare A vector, which is used to update the swarming value.
00026  */
00027 ISR(TIMER1_COMPA_vect)
00028 {
00029     /* Enable the flag to update the swarming values. */
00030     Swarming_Enable_Flag();
00031 }
00032
00033 /**
00034  * @brief Interrupt for timer 2 compare A vector, which is used to update the state machine.
00035  */
00036 ISR(TIMER2_COMPA_vect)
00037 {
00038     /* Enable the flag to update the state machine. */
00039     State_Machine_Enable_Flag();
00040 }
  
```

3.44 Microcontroller/Interrupt/Interrupt.h File Reference

This graph shows which files directly or indirectly include this file:



3.45 Interrupt.h

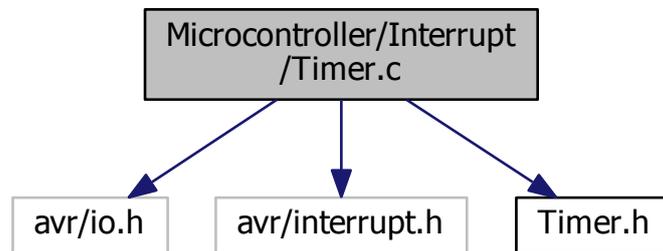
```
00001 /**
00002  * @file Interrupt.c
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 1/29/2015\n Created: 1/27/2015 4:28:45 PM
00006  * @brief This file is code for the interrupts on the ATmega328P.
00007  */
00008
00009 #ifndef INTERRUPT_H_
00010 #define INTERRUPT_H_
00011
00012 #endif /* INTERRUPT_H_ */
```

3.46 Microcontroller/Interrupt/Timer.c File Reference

This file is code to control the Timers on the ATmega328P.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include "Timer.h"
```

Include dependency graph for Timer.c:



Functions

- void [Timer0_Initialize](#) ()
Initializes Timer0 with a period of 20ms.
- void [Timer0_Enable](#) ()
Enable Timer0 interrupt flag.
- void [Timer0_Disable](#) ()
Disable Timer0 interrupt flag.
- void [Timer1_Initialize](#) ()
Initializes Timer1 as an interrupt at a period of ?.
- void [Timer1_Enable](#) ()
Enable Timer1 interrupt flag.
- void [Timer1_Disable](#) ()
Disable Timer1 interrupt flag.
- void [Timer2_Initialize](#) ()
Initializes Timer2 with a period of ?.
- void [Timer2_Enable](#) ()
Enable Timer2 interrupt flag.
- void [Timer2_Disable](#) ()
Disable Timer2 interrupt flag.
- void [Timer_Initialize](#) (void)
Initializes the timers that are used in the project.

3.46.1 Detailed Description

This file is code to control the Timers on the ATmega328P.

Author

Nicholas Sikkema

Version

Revision: 2.0

Date

Last Updated: 4/21/2015 PM
Created: 11/12/2014

Definition in file [Timer.c](#).

3.46.2 Function Documentation

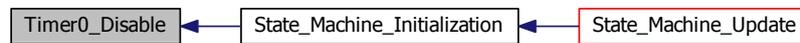
3.46.2.1 void Timer0_Disable (void)

Disable Timer0 interrupt flag.

Definition at line 40 of file [Timer.c](#).

Referenced by [State_Machine_Initialization\(\)](#).

Here is the caller graph for this function:



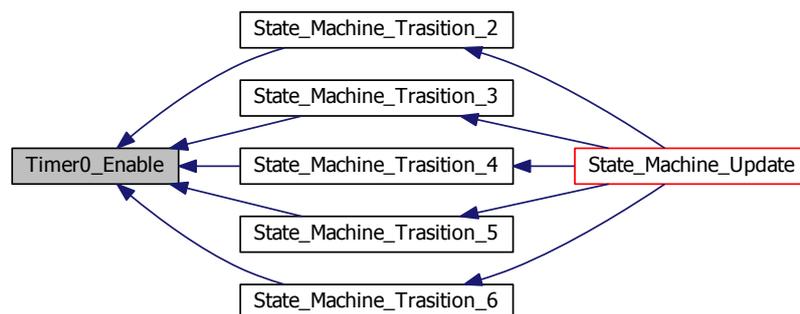
3.46.2.2 void Timer0_Enable (void)

Enable Timer0 interrupt flag.

Definition at line 31 of file [Timer.c](#).

Referenced by [State_Machine_Trasition_2\(\)](#), [State_Machine_Trasition_3\(\)](#), [State_Machine_Trasition_4\(\)](#), [State_Machine_Trasition_5\(\)](#), and [State_Machine_Trasition_6\(\)](#).

Here is the caller graph for this function:



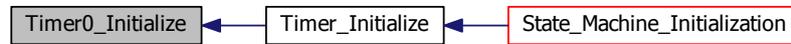
3.46.2.3 void Timer0_Initialize () [inline]

Initializes Timer0 with a period of 20ms.

Definition at line 16 of file [Timer.c](#).

Referenced by [Timer_Initialize\(\)](#).

Here is the caller graph for this function:



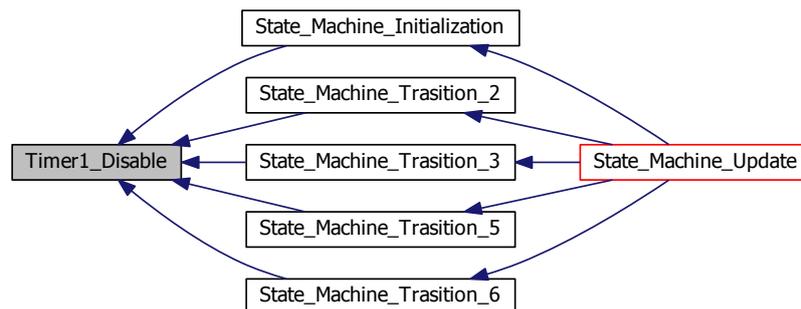
3.46.2.4 void Timer1_Disable (void)

Disable Timer1 interrupt flag.

Definition at line 73 of file [Timer.c](#).

Referenced by [State_Machine_Initialization\(\)](#), [State_Machine_Trasition_2\(\)](#), [State_Machine_Trasition_3\(\)](#), [State_Machine_Trasition_5\(\)](#), and [State_Machine_Trasition_6\(\)](#).

Here is the caller graph for this function:



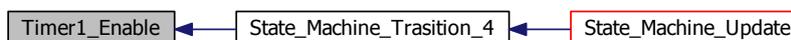
3.46.2.5 void Timer1_Enable (void)

Enable Timer1 interrupt flag.

Definition at line 64 of file [Timer.c](#).

Referenced by [State_Machine_Trasition_4\(\)](#).

Here is the caller graph for this function:



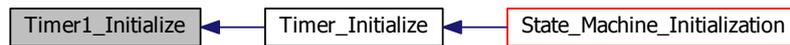
3.46.2.6 void Timer1_Initialize () [inline]

Initializes Timer1 as an interrupt at a period of ?.

Definition at line 49 of file [Timer.c](#).

Referenced by [Timer_Initialize\(\)](#).

Here is the caller graph for this function:



3.46.2.7 void Timer2_Disable (void)

Disable Timer2 interrupt flag.

Definition at line 107 of file [Timer.c](#).

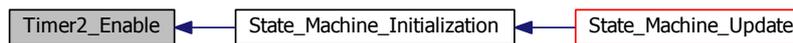
3.46.2.8 void Timer2_Enable (void)

Enable Timer2 interrupt flag.

Definition at line 98 of file [Timer.c](#).

Referenced by [State_Machine_Initialization\(\)](#).

Here is the caller graph for this function:



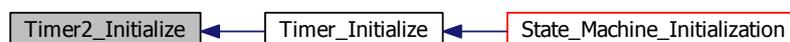
3.46.2.9 void Timer2_Initialize () [inline]

Initializes Timer2 with a period of ?.

Definition at line 83 of file [Timer.c](#).

Referenced by [Timer_Initialize\(\)](#).

Here is the caller graph for this function:



3.46.2.10 void Timer_Initialize (void)

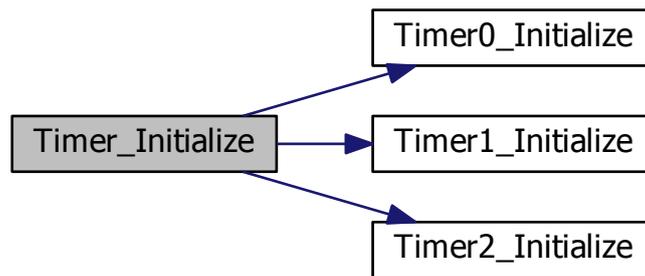
Initializes the timers that are used in the project.

Definition at line 116 of file [Timer.c](#).

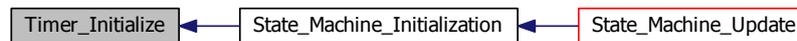
References [Timer0_Initialize\(\)](#), [Timer1_Initialize\(\)](#), and [Timer2_Initialize\(\)](#).

Referenced by [State_Machine_Initialization\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.47 Timer.c

```

00001 /**
00002  * @file Timer.c
00003  * @author Nicholas Sikkema
00004  * @version Revision: 2.0
00005  * @date Last Updated: 4/21/2015 PM\n Created: 11/12/2014
00006  * @brief This file is code to control the Timers on the ATmega328P.
00007  */
00008
00009 #include <avr/io.h>
00010 #include <avr/interrupt.h>
00011 #include "Timer.h"
00012
00013 /**
00014  * @brief Initializes Timer0 with a period of 20ms.
00015  */
00016 inline void Timer0_Initialize ()
00017 {
00018     /* Set Timer0 initial counter value to 0. */
00019     TCNT0 = 0x00;
00020     /* Set timer period of 12.5 ms. */
00021     OCR0A = 0xC2;
00022     /* Set the output to be in CTC mode. */
00023     TCCR0A |= (1<<WGM01);
00024     /* Set the timer prescaler to 1024. */
00025     TCCR0B |= (1<<CS02)|(1<<CS00);
00026 }
00027
  
```

```

00028 /**
00029  * @brief Enable Timer0 interrupt flag.
00030  */
00031 void Timer0_Enable()
00032 {
00033     /* Set the Timer0A Interrupt flag to 1. */
00034     TIMSK0 |= (1<<OCIE0A);
00035 }
00036
00037 /**
00038  * @brief Disable Timer0 interrupt flag.
00039  */
00040 void Timer0_Disable()
00041 {
00042     /* Set the Timer0A Interrupt flag to 0. */
00043     TIMSK0 &= ~(1<<OCIE0A);
00044 }
00045
00046 /**
00047  * @brief Initializes Timer1 as an interrupt at a period of ?.
00048  */
00049 inline void Timer1_Initialize ()
00050 {
00051     /* Set Timer0 initial counter value to 0. */
00052     TCNT1 = 0x00;
00053     /* Set timer period of 25 ms. */
00054     OCR1A = 0x0186;
00055     /* Set the output to be in CTC mode. */
00056     TCCR1A |= (1<<WGM12);
00057     /* Set the timer prescaler to 1024. */
00058     TCCR1B |= (1<<CS12)|(1<<CS10);
00059 }
00060
00061 /**
00062  * @brief Enable Timer1 interrupt flag.
00063  */
00064 void Timer1_Enable()
00065 {
00066     /* Set the Timer1A Interrupt flag to 1. */
00067     TIMSK1 |= (1<<OCIE1A);
00068 }
00069
00070 /**
00071  * @brief Disable Timer1 interrupt flag.
00072  */
00073 void Timer1_Disable()
00074 {
00075     /* Set the Timer1A Interrupt flag to 0. */
00076     TIMSK1 &= ~(1<<OCIE1A);
00077 }
00078
00079
00080 /**
00081  * @brief Initializes Timer2 with a period of ?.
00082  */
00083 inline void Timer2_Initialize ()
00084 {
00085     /* Set Timer0 initial counter value to 0. */
00086     TCNT2 = 0x00;
00087     /* Set timer period of 12.5 ms. */
00088     OCR2A = 0xC2;
00089     /* Set the output to be in CTC mode. */
00090     TCCR2A |= (1<<WGM21);
00091     /* Set the timer prescaler to 1024. */
00092     TCCR2B |= (1<<CS22)|(1<<CS20);
00093 }
00094
00095 /**
00096  * @brief Enable Timer2 interrupt flag.
00097  */
00098 void Timer2_Enable()
00099 {
00100     /* Set the Timer2A interrupt flag to 1. */
00101     TIMSK2 |= (1<<OCIE2A);
00102 }
00103
00104 /**
00105  * @brief Disable Timer2 interrupt flag.
00106  */
00107 void Timer2_Disable()
00108 {
00109     /* Set the Timer2A interrupt flag to 0. */
00110     TIMSK2 &= ~(1<<OCIE2A);
00111 }
00112
00113 /**
00114  * @brief Initializes the timers that are used in the project.

```

```

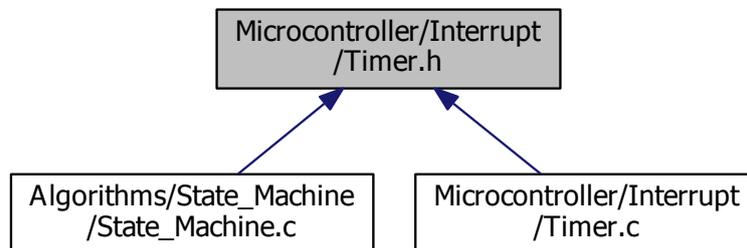
00115  */
00116 void Timer_Initialize(void)
00117 {
00118     /* Initialize the timers. */
00119     Timer0_Initialize();
00120     Timer1_Initialize();
00121     Timer2_Initialize();
00122     /* Enable global interrupts. */
00123     sei();
00124 }

```

3.48 Microcontroller/Interrupt/Timer.h File Reference

This file is code to control the Timers on the ATmega328P.

This graph shows which files directly or indirectly include this file:



Functions

- void [Timer_Initialize](#) (void)
Initializes the timers that are used in the project.
- void [Timer0_Enable](#) (void)
Enable Timer0 interrupt flag.
- void [Timer0_Disable](#) (void)
Disable Timer0 interrupt flag.
- void [Timer1_Enable](#) (void)
Enable Timer1 interrupt flag.
- void [Timer1_Disable](#) (void)
Disable Timer1 interrupt flag.
- void [Timer2_Enable](#) (void)
Enable Timer2 interrupt flag.
- void [Timer2_Disable](#) (void)
Disable Timer2 interrupt flag.

3.48.1 Detailed Description

This file is code to control the Timers on the ATmega328P.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 2/12/2015

Created:

Definition in file [Timer.h](#).

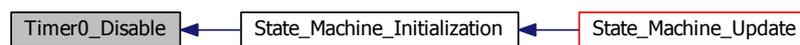
3.48.2 Function Documentation

3.48.2.1 void Timer0_Disable (void)

Disable Timer0 interrupt flag.

Definition at line 40 of file [Timer.c](#).Referenced by [State_Machine_Initialization\(\)](#).

Here is the caller graph for this function:

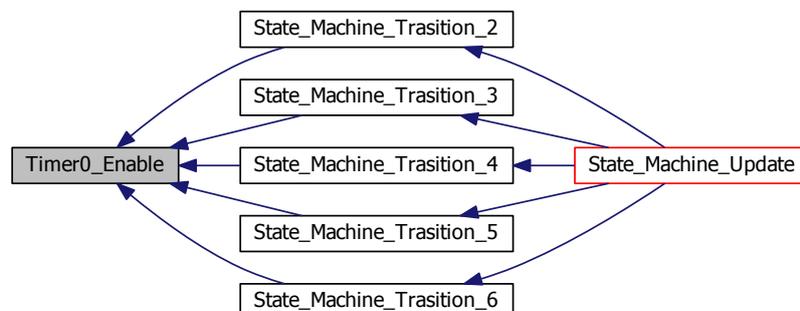


3.48.2.2 void Timer0_Enable (void)

Enable Timer0 interrupt flag.

Definition at line 31 of file [Timer.c](#).Referenced by [State_Machine_Trasition_2\(\)](#), [State_Machine_Trasition_3\(\)](#), [State_Machine_Trasition_4\(\)](#), [State_Machine_Trasition_5\(\)](#), and [State_Machine_Trasition_6\(\)](#).

Here is the caller graph for this function:



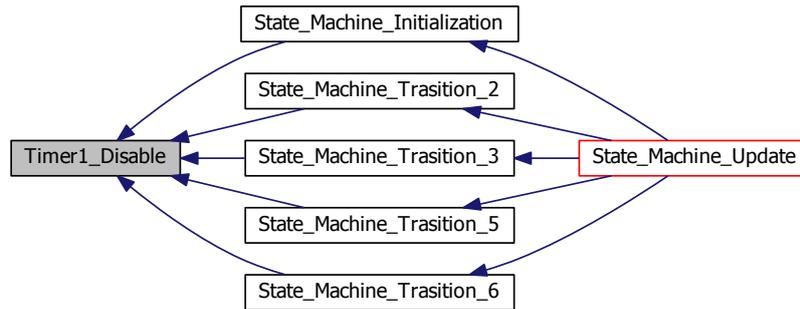
3.48.2.3 void Timer1_Disable (void)

Disable Timer1 interrupt flag.

Definition at line 73 of file [Timer.c](#).

Referenced by [State_Machine_Initialization\(\)](#), [State_Machine_Trasition_2\(\)](#), [State_Machine_Trasition_3\(\)](#), [State_Machine_Trasition_5\(\)](#), and [State_Machine_Trasition_6\(\)](#).

Here is the caller graph for this function:



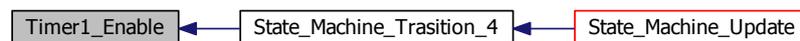
3.48.2.4 void Timer1_Enable (void)

Enable Timer1 interrupt flag.

Definition at line 64 of file [Timer.c](#).

Referenced by [State_Machine_Trasition_4\(\)](#).

Here is the caller graph for this function:



3.48.2.5 void Timer2_Disable (void)

Disable Timer2 interrupt flag.

Definition at line 107 of file [Timer.c](#).

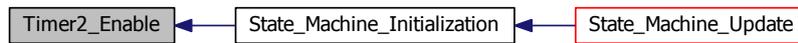
3.48.2.6 void Timer2_Enable (void)

Enable Timer2 interrupt flag.

Definition at line 98 of file [Timer.c](#).

Referenced by [State_Machine_Initialization\(\)](#).

Here is the caller graph for this function:



3.48.2.7 void Timer_Initialize (void)

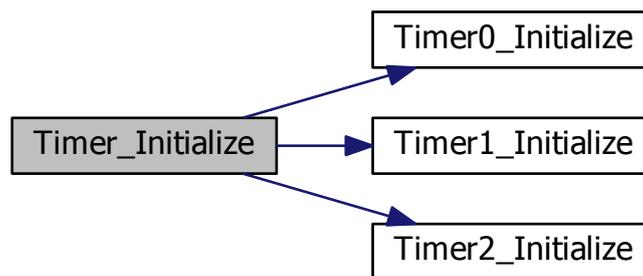
Initializes the timers that are used in the project.

Definition at line 116 of file [Timer.c](#).

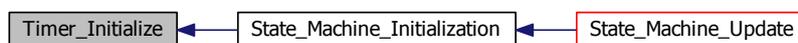
References [Timer0_Initialize\(\)](#), [Timer1_Initialize\(\)](#), and [Timer2_Initialize\(\)](#).

Referenced by [State_Machine_Initialization\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.49 Timer.h

```

00001 /**
00002  * @file Timer.h
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 2/12/2015\n Created:
00006  * @brief This file is code to control the Timers on the ATmega328P.
00007  */
00008
00009 #ifndef Timer_H_
00010 #define Timer_H_
  
```

```

00011
00012 void Timer_Initialize(void);
00013 void Timer0_Enable(void);
00014 void Timer0_Disable(void);
00015 void Timer1_Enable(void);
00016 void Timer1_Disable(void);
00017 void Timer2_Enable(void);
00018 void Timer2_Disable(void);
00019
00020 #endif

```

3.50 Subsystem/H_Bridge/DRV8830.c File Reference

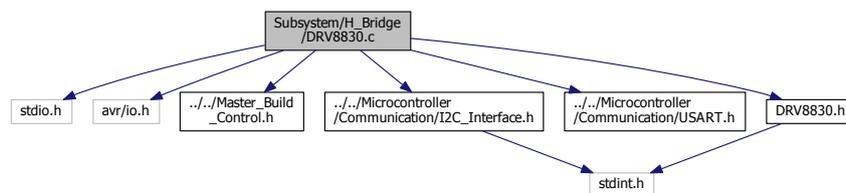
This file is code to read control the h-bridges.

```

#include <stdio.h>
#include <avr/io.h>
#include "../Master_Build_Control.h"
#include "../Microcontroller/Communication/I2C_Interface.h"
#include "../Microcontroller/Communication/USART.h"
#include "DRV8830.h"

```

Include dependency graph for DRV8830.c:



Macros

- `#define H_Bridge_Voltage_Conversion 12.5`
The conversion factor used between the floating point value and the value for the h-bridge.
- `#define H_Bridge_Max_Voltage_VSET (uint8_t)(H_Bridge_Voltage_Conversion*Motor_Max_Voltage)`
The h-bridge value that is equivalent to Motor_Max_Voltage.
- `#define H_Bridge_Min_Voltage 0.48`
The minimum voltage that the h-bridge will allow.
- `#define H_Bridge_Min_Voltage_VSET (uint8_t)(H_Bridge_Voltage_Conversion*H_Bridge_Min_Voltage)`
The h-bridge value that is equivalent to Motor_Max_Voltage.
- `#define Clear_Fault_Bit 7`
The clear fault bit for the h-bridge.
- `#define Motor_Fault_Count_Max 5`
Maximum amount of times allowed to fault before sending that the motor failed.
- `#define Motor_Max_Jump 25`
Maximum voltage jump allowed for a motor.

Enumerations

- enum `Motor_Adress` { `Motor_Address_X` = 0x60, `Motor_Address_Y` = 0x62, `Motor_Address_Z` = 0x66 }
I2C h-bridge addresses for the motors.
- enum `DRV8830_Registers` { `DRV8830_Control` = 0x00, `DRV8830_Fault` = 0x01 }

The register addresses for the DRV8830.

- enum [DRV8830_Modes](#) { [DRV8830_Standby](#) = 0, [DRV8830_Reverse](#) = 1, [DRV8830_Forward](#) = 2, [DRV8830_Stop](#) = 3 }

The list of motor modes for the DRV8830.

Functions

- [uint8_t DRV8830_Voltage_To_Int](#) (float Voltage)

This function converts a desired voltage to the equivalent voltage value for the h-bridge.

- [uint8_t DRV8830_Jump_Check](#) ([uint8_t *New_Voltage](#), [uint8_t *New_Mode](#), [uint8_t Old_Voltage](#), [uint8_t Old_Mode](#))

This function checks the desired equivalent voltage value. Checking against the previous mode and voltage, to make sure of three conditions. First is that the new voltage is not significantly greater than the previous voltage. Second is to set the flags for the start-up of the motor. This will be used to make sure that not more one motor is being started up at the same time. Third is to prevent the motors from switching polarity too quickly.

- [uint8_t DRV8830_Mode](#) (float *Voltage)

This function uses the a desired voltage to determine the mode for the h-bridge.

- [uint8_t DRV8830_Start_Up_Flag_Control](#) ([uint8_t Motor_Address](#))

This function checks whether or not a motor should be updated. Making sure that there are no motors that are turning on at the same time.

- [void DRV8830_Set_Voltage](#) ([uint8_t Address](#), [uint8_t Voltage_Value](#), [uint8_t Mode](#))

This function sets the output voltage for the h-bridge.

- [void DRV8830_Set_Voltage_Motor_X](#) (float Voltage)

This function sets the output voltage for the X-axis h-bridge. Note that this function only Updates the h-bridge if the mode or voltage is different.

- [void DRV8830_Set_Voltage_Motor_Y](#) (float Voltage)

This function sets the output voltage for the Y-axis h-bridge. Note that this function only Updates the h-bridge if the mode or voltage is different.

- [void DRV8830_Set_Voltage_Motor_Z](#) (float Voltage)

This function sets the output voltage for the Z-axis h-bridge. Note that this function only Updates the h-bridge if the mode or voltage is different.

- [void DRV8830_Initialize](#) (void)

This function sets the initial output voltage for all three h-bridges.

- [uint8_t DRV8830_Read_Fault](#) ([uint8_t Address](#))

This function reads the fault byte for the h-bridge.

- [uint8_t DRV8830_Read_Fault_Motor_X](#) (void)

This function reads the fault byte for the X-axis h-bridge.

- [uint8_t DRV8830_Read_Fault_Motor_Y](#) (void)

This function reads the fault byte for the Y-axis h-bridge.

- [uint8_t DRV8830_Read_Fault_Motor_Z](#) (void)

This function reads the fault byte for the Z-axis h-bridge.

- [void DRV8830_Clear_Fault](#) ([uint8_t Address](#))

This function clears the fault byte for the h-bridge.

- [void DRV8830_Clear_Fault_Motor_X](#) (void)

This function clears the fault byte for the X-axis h-bridge.

- [void DRV8830_Clear_Fault_Motor_Y](#) (void)

This function clears the fault byte for the Y-axis h-bridge.

- [void DRV8830_Clear_Fault_Motor_Z](#) (void)

This function clears the fault byte for the Z-axis h-bridge.

- [void DRV8830_Reset_Motor_X](#) (void)

- [void DRV8830_Reset_Motor_Y](#) (void)

- [void DRV8830_Reset_Motor_Z](#) (void)

- `uint8_t DRV8830_Check_Motor_X` (void)
This function checks to see if the X-axis motor is failing. Counting and resetting the motor faults to make sure the motor is able to continue.
- `uint8_t DRV8830_Check_Motor_Y` (void)
This function checks to see if the Y-axis motor is failing. Counting and resetting the motor faults to make sure the motor is able to continue.
- `uint8_t DRV8830_Check_Motor_Z` (void)
This function checks to see if the Z-axis motor is failing. Counting and resetting the motor faults to make sure the motor is able to continue.
- `uint8_t DRV8830_Check_Motors` (void)
This function checks to see if any of the 3 motors is failing.

3.50.1 Detailed Description

This file is code to read control the h-bridges.

Author

Nicholas Sikkema

Version

Revision: 2.0

Date

Last Updated: 4/21/2015

Created: 11/8/2014 3:54:57 PM

Definition in file [DRV8830.c](#).

3.50.2 Macro Definition Documentation

3.50.2.1 `#define Clear_Fault_Bit 7`

The clear fault bit for the h-bridge.

Definition at line 18 of file [DRV8830.c](#).

Referenced by [DRV8830_Clear_Fault\(\)](#).

3.50.2.2 `#define H_Bridge_Max_Voltage_VSET (uint8_t)(H_Bridge_Voltage_Conversion*Motor_Max_Voltage)`

The h-bridge value that is equivalent to `Motor_Max_Voltage`.

Definition at line 12 of file [DRV8830.c](#).

Referenced by [DRV8830_Voltage_To_Int\(\)](#).

3.50.2.3 `#define H_Bridge_Min_Voltage 0.48`

The minimum voltage that the h-bridge will allow.

Definition at line 14 of file [DRV8830.c](#).

Referenced by [DRV8830_Mode\(\)](#), and [DRV8830_Voltage_To_Int\(\)](#).

3.50.2.4 `#define H_Bridge_Min_Voltage_VSET (uint8_t)(H_Bridge_Voltage_Conversion*H_Bridge_Min_Voltage)`

The h-bridge value that is equivalent to `Motor_Max_Voltage`.

Definition at line 16 of file [DRV8830.c](#).

Referenced by [DRV8830_Jump_Check\(\)](#).

3.50.2.5 `#define H_Bridge_Voltage_Conversion 12.5`

The conversion factor used between the floating point value and the value for the h-bridge.

Definition at line 10 of file [DRV8830.c](#).

Referenced by [DRV8830_Voltage_To_Int\(\)](#).

3.50.2.6 `#define Motor_Fault_Count_Max 5`

Maximum amount of times allowed to fault before sending that the motor failed.

Definition at line 20 of file [DRV8830.c](#).

Referenced by [DRV8830_Check_Motor_X\(\)](#), [DRV8830_Check_Motor_Y\(\)](#), and [DRV8830_Check_Motor_Z\(\)](#).

3.50.2.7 `#define Motor_Max_Jump 25`

Maximum voltage jump allowed for a motor.

Definition at line 22 of file [DRV8830.c](#).

Referenced by [DRV8830_Jump_Check\(\)](#).

3.50.3 Enumeration Type Documentation

3.50.3.1 `enum DRV8830_Modes`

The list of motor modes for the DRV8830.

Enumerator

DRV8830_Standby

DRV8830_Reverse

DRV8830_Forward

DRV8830_Stop

Definition at line 40 of file [DRV8830.c](#).

3.50.3.2 `enum DRV8830_Registers`

The register addresses for the DRV8830.

Enumerator

DRV8830_Control

DRV8830_Fault

Definition at line 33 of file [DRV8830.c](#).

3.50.3.3 enum Motor_Adress

I2C h-bridge addresses for the motors.

Enumerator

Motor_Address_X

Motor_Address_Y

Motor_Address_Z

Definition at line 25 of file [DRV8830.c](#).

3.50.4 Function Documentation

3.50.4.1 uint8_t DRV8830_Check_Motor_X (void)

This function checks to see if the X-axis motor is failing. Counting and resetting the motor faults to make sure the motor is able to continue.

Returns

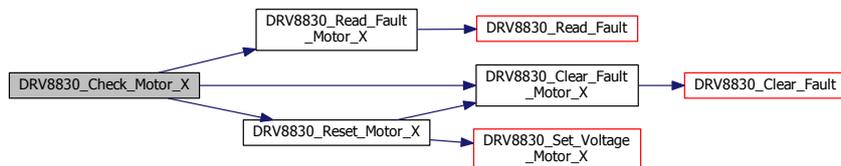
Whether or not the fault count is greater than a specified amount.

Definition at line 605 of file [DRV8830.c](#).

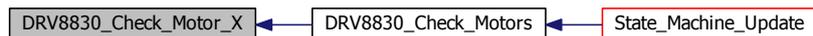
References [DRV8830_Clear_Fault_Motor_X\(\)](#), [DRV8830_Read_Fault_Motor_X\(\)](#), [DRV8830_Reset_Motor_X\(\)](#), and [Motor_Fault_Count_Max](#).

Referenced by [DRV8830_Check_Motors\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.50.4.2 uint8_t DRV8830_Check_Motor_Y (void)

This function checks to see if the Y-axis motor is failing. Counting and resetting the motor faults to make sure the motor is able to continue.

Returns

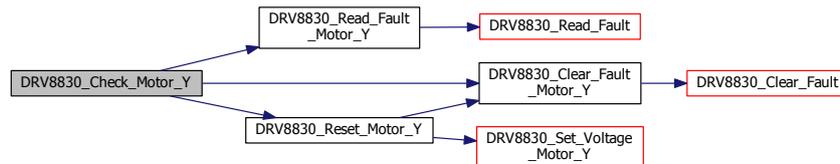
Whether or not the fault count is greater than a specified amount.

Definition at line 637 of file [DRV8830.c](#).

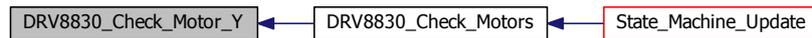
References [DRV8830_Clear_Fault_Motor_Y\(\)](#), [DRV8830_Read_Fault_Motor_Y\(\)](#), [DRV8830_Reset_Motor_Y\(\)](#), and [Motor_Fault_Count_Max](#).

Referenced by [DRV8830_Check_Motors\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.50.4.3 uint8_t DRV8830_Check_Motor_Z (void)

This function checks to see if the Z-axis motor is failing. Counting and resetting the motor faults to make sure the motor is able to continue.

Returns

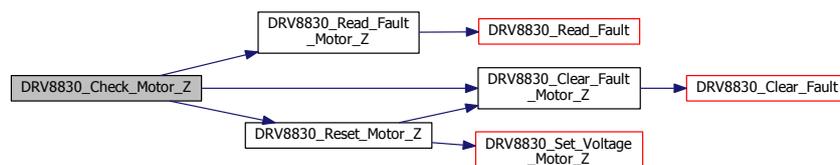
Whether or not the fault count is greater than a specified amount.

Definition at line 669 of file [DRV8830.c](#).

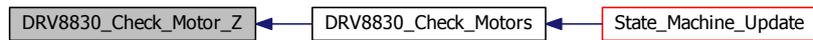
References [DRV8830_Clear_Fault_Motor_Z\(\)](#), [DRV8830_Read_Fault_Motor_Z\(\)](#), [DRV8830_Reset_Motor_Z\(\)](#), and [Motor_Fault_Count_Max](#).

Referenced by [DRV8830_Check_Motors\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.50.4.4 uint8_t DRV8830_Check_Motors (void)

This function checks to see if any of the 3 motors is failing.

Returns

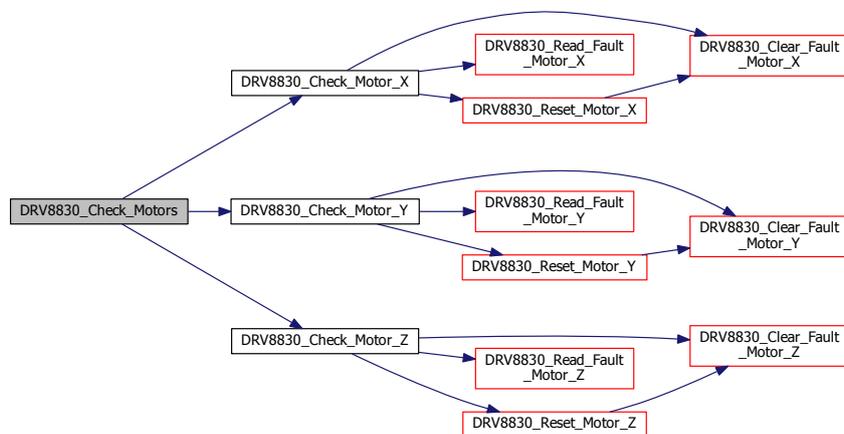
- Whether or not the fault count is greater than a specified amount for each motor.
- Bit 0: Z-axis motor
- Bit 1: Y-axis motor
- Bit 2: X-axis motor

Definition at line 703 of file [DRV8830.c](#).

References [DRV8830_Check_Motor_X\(\)](#), [DRV8830_Check_Motor_Y\(\)](#), and [DRV8830_Check_Motor_Z\(\)](#).

Referenced by [State_Machine_Update\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.50.4.5 `void DRV8830_Clear_Fault (uint8_t Address) [inline]`

This function clears the fault byte for the h-bridge.

Parameters

<i>Address</i>	The h-bridge I2C address.
----------------	---------------------------

Definition at line 542 of file [DRV8830.c](#).

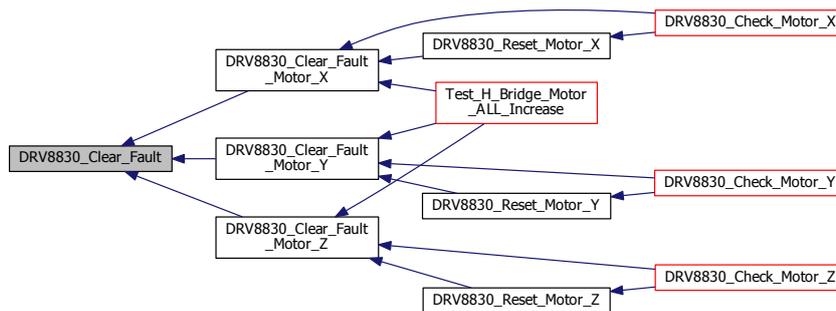
References [Clear_Fault_Bit](#), [DRV8830_Fault](#), and [I2C_Interface_Write\(\)](#).

Referenced by [DRV8830_Clear_Fault_Motor_X\(\)](#), [DRV8830_Clear_Fault_Motor_Y\(\)](#), and [DRV8830_Clear_Fault_Motor_Z\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.50.4.6 void DRV8830_Clear_Fault_Motor_X(void)

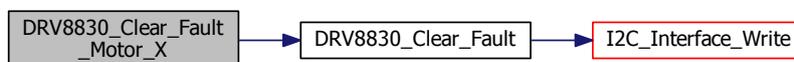
This function clears the fault byte for the X-axis h-bridge.

Definition at line 551 of file [DRV8830.c](#).

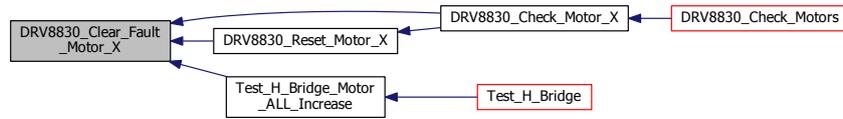
References [DRV8830_Clear_Fault\(\)](#), and [Motor_Address_X](#).

Referenced by [DRV8830_Check_Motor_X\(\)](#), [DRV8830_Reset_Motor_X\(\)](#), and [Test_H_Bridge_Motor_ALL_Increase\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.50.4.7 void DRV8830_Clear_Fault_Motor_Y (void)

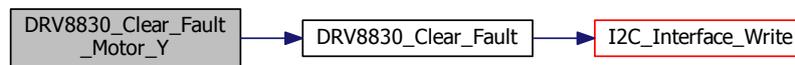
This function clears the fault byte for the Y-axis h-bridge.

Definition at line 560 of file [DRV8830.c](#).

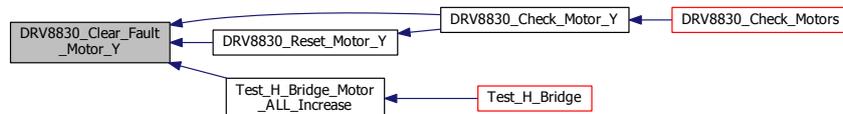
References [DRV8830_Clear_Fault\(\)](#), and [Motor_Address_Y](#).

Referenced by [DRV8830_Check_Motor_Y\(\)](#), [DRV8830_Reset_Motor_Y\(\)](#), and [Test_H_Bridge_Motor_ALL_Increase\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.50.4.8 void DRV8830_Clear_Fault_Motor_Z (void)

This function clears the fault byte for the Z-axis h-bridge.

Definition at line 569 of file [DRV8830.c](#).

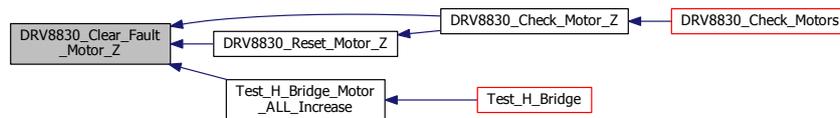
References [DRV8830_Clear_Fault\(\)](#), and [Motor_Address_Z](#).

Referenced by [DRV8830_Check_Motor_Z\(\)](#), [DRV8830_Reset_Motor_Z\(\)](#), and [Test_H_Bridge_Motor_ALL_Increase\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.50.4.9 void DRV8830_Initialize (void) [inline]

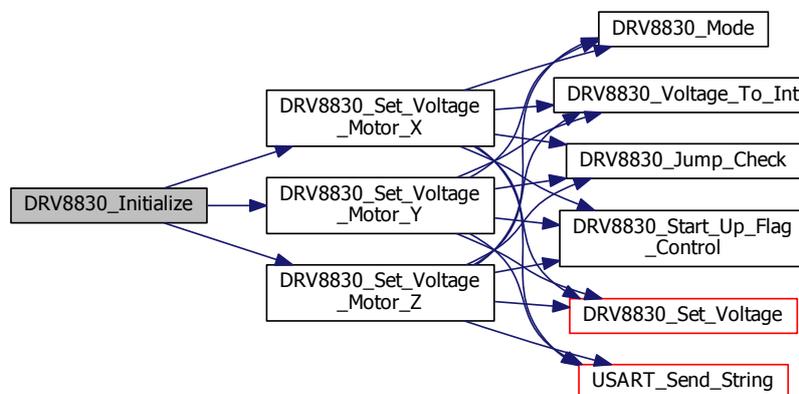
This function sets the initial output voltage for all three h-bridges.

Definition at line 487 of file [DRV8830.c](#).

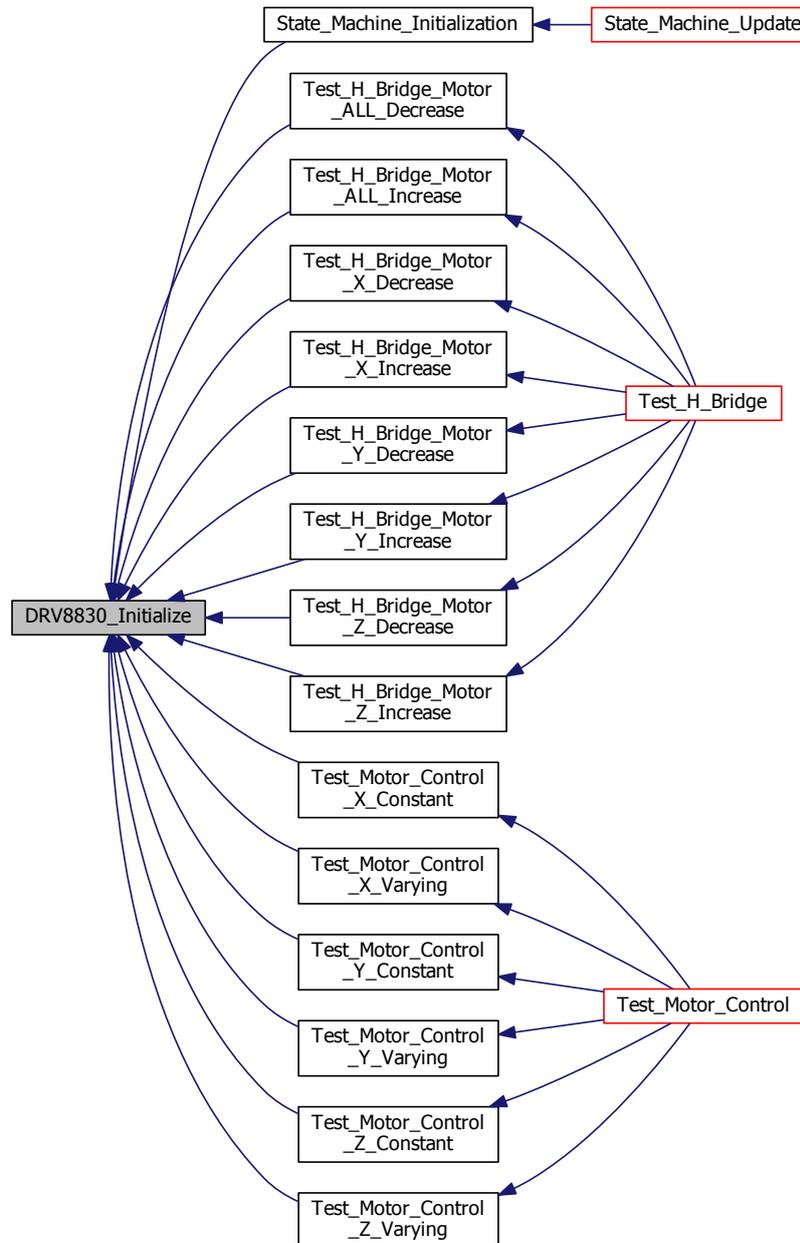
References [DRV8830_Set_Voltage_Motor_X\(\)](#), [DRV8830_Set_Voltage_Motor_Y\(\)](#), and [DRV8830_Set_Voltage_Motor_Z\(\)](#).

Referenced by [State_Machine_Initialization\(\)](#), [Test_H_Bridge_Motor_ALL_Decrease\(\)](#), [Test_H_Bridge_Motor_ALL_Increase\(\)](#), [Test_H_Bridge_Motor_X_Decrease\(\)](#), [Test_H_Bridge_Motor_X_Increase\(\)](#), [Test_H_Bridge_Motor_Y_Decrease\(\)](#), [Test_H_Bridge_Motor_Y_Increase\(\)](#), [Test_H_Bridge_Motor_Z_Decrease\(\)](#), [Test_H_Bridge_Motor_Z_Increase\(\)](#), [Test_Motor_Control_X_Constant\(\)](#), [Test_Motor_Control_X_Varying\(\)](#), [Test_Motor_Control_Y_Constant\(\)](#), [Test_Motor_Control_Y_Varying\(\)](#), [Test_Motor_Control_Z_Constant\(\)](#), and [Test_Motor_Control_Z_Varying\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



```

3.50.4.10 uint8_t DRV8830_Jump_Check ( uint8_t * New_Voltage, uint8_t * New_Mode, uint8_t Old_Voltage, uint8_t Old_Mode
) [inline]

```

This function checks the desired equivalent voltage value. Checking against the previous mode and voltage, to make sure of three conditions. First is that the new voltage is not significantly greater than the previous voltage. Second is to set the flags for the start-up of the motor. This will be used to make sure that not more one motor is being started up at the same time. Third is to prevent the motors from switching polarity too quickly.

Parameters

<i>New_Voltage</i>	The desired equivalent voltage value of the h-bridge. Note that the function updates this variable with a new voltage if it matches on of the three conditions.
<i>New_Mode</i>	The desired mode of the h-bridge. Note that the function updates this variable with a new mode if it matches on of the three conditions.
<i>Old_Voltage</i>	The current equivalent voltage value of the h-bridge.
<i>Old_Mode</i>	The mode of the h-bridge.

Returns

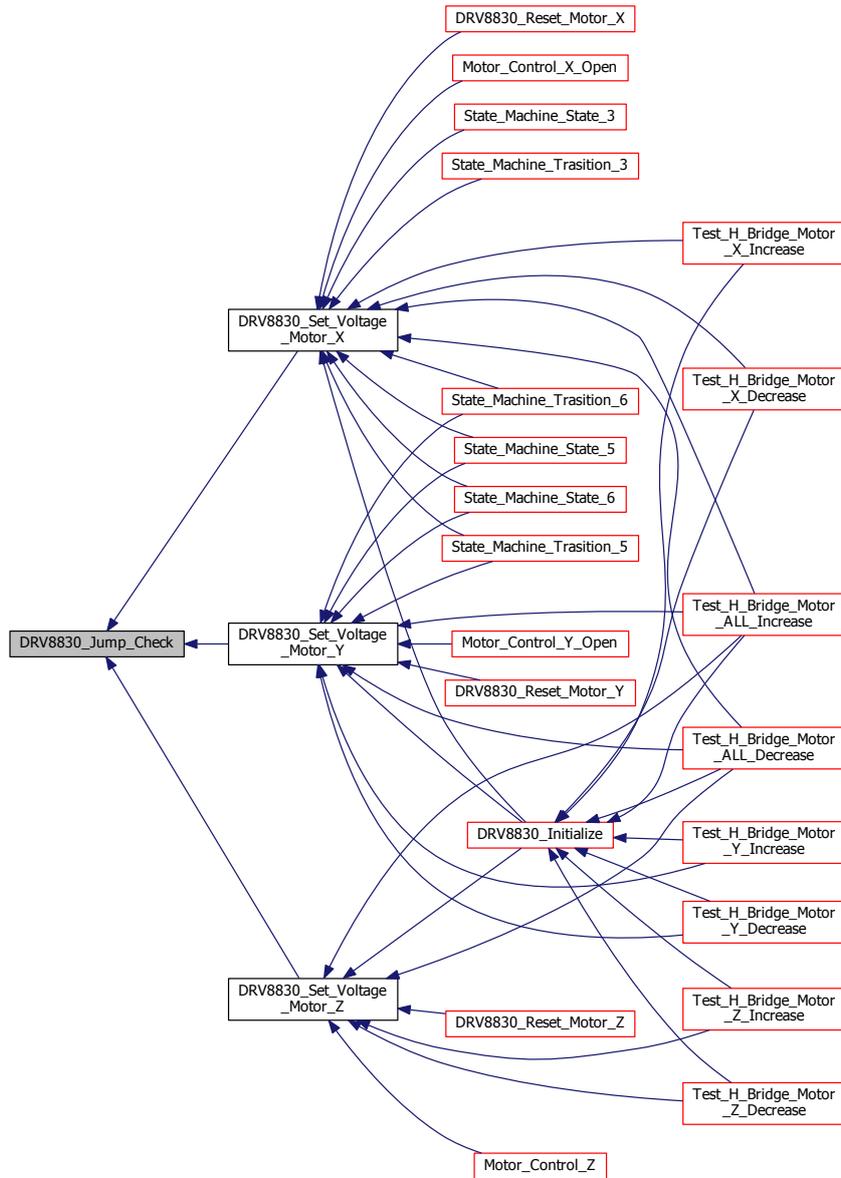
The start-up flag information for the motor.

Definition at line 112 of file [DRV8830.c](#).

References [DRV8830_Forward](#), [DRV8830_Reverse](#), [DRV8830_Stop](#), [H_Bridge_Min_Voltage_VSET](#), and [Motor↔_Max_Jump](#).

Referenced by [DRV8830_Set_Voltage_Motor_X\(\)](#), [DRV8830_Set_Voltage_Motor_Y\(\)](#), and [DRV8830_Set_↔_Voltage_Motor_Z\(\)](#).

Here is the caller graph for this function:



3.50.4.11 `uint8_t DRV8830_Mode (float * Voltage) [inline]`

This function uses the a desired voltage to determine the mode for the h-bridge.

Parameters

<i>Voltage</i>	The desired voltage for the motor to be at.
----------------	---

Returns

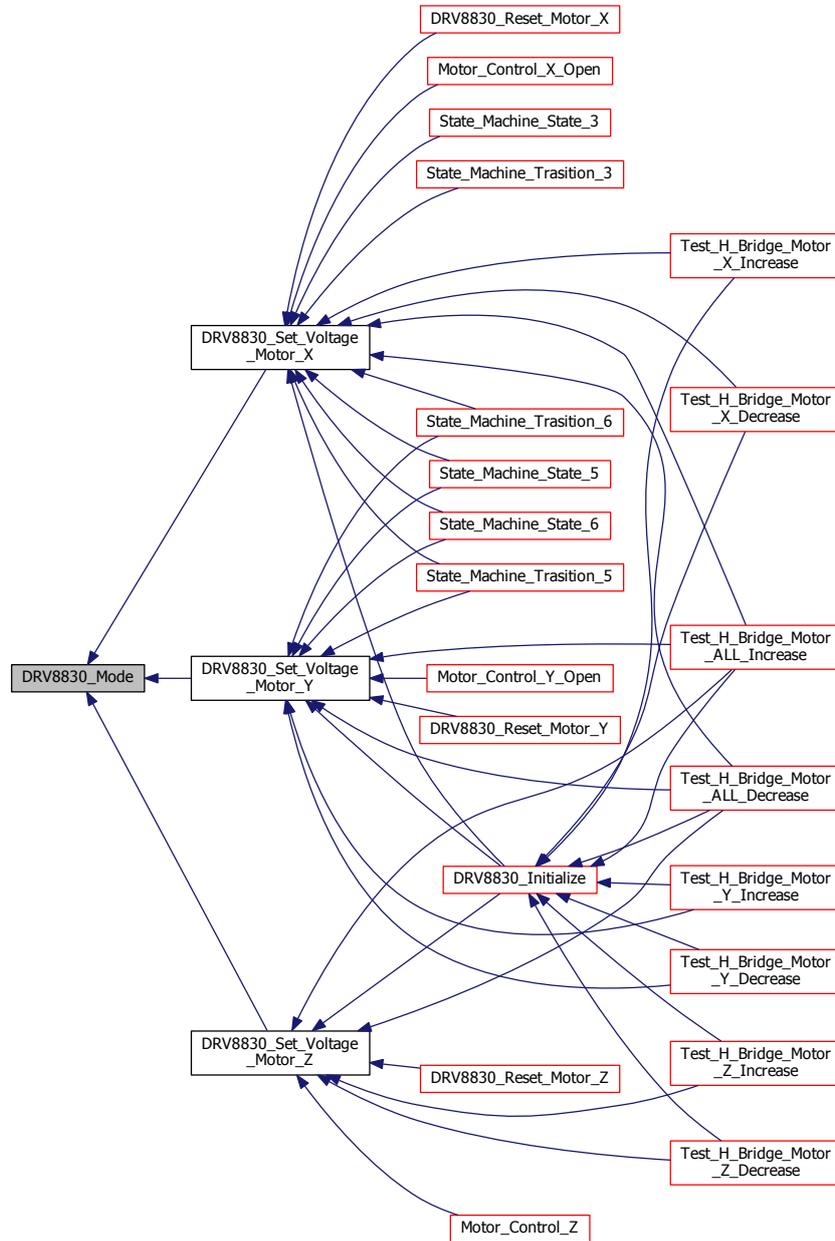
The new mode for the h-bridge.

Definition at line 234 of file [DRV8830.c](#).

References [DRV8830_Forward](#), [DRV8830_Reverse](#), [DRV8830_Stop](#), and [H_Bridge_Min_Voltage](#).

Referenced by [DRV8830_Set_Voltage_Motor_X\(\)](#), [DRV8830_Set_Voltage_Motor_Y\(\)](#), and [DRV8830_Set_Voltage_Motor_Z\(\)](#).

Here is the caller graph for this function:



3.50.4.12 uint8_t DRV8830_Read_Fault (uint8_t Address) [inline]

This function reads the fault byte for the h-bridge.

Parameters

<i>Address</i>	The h-bridge I2C address.
----------------	---------------------------

Returns

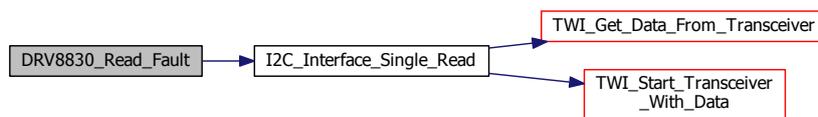
The current fault byte for the h-bridge.

Definition at line 502 of file [DRV8830.c](#).

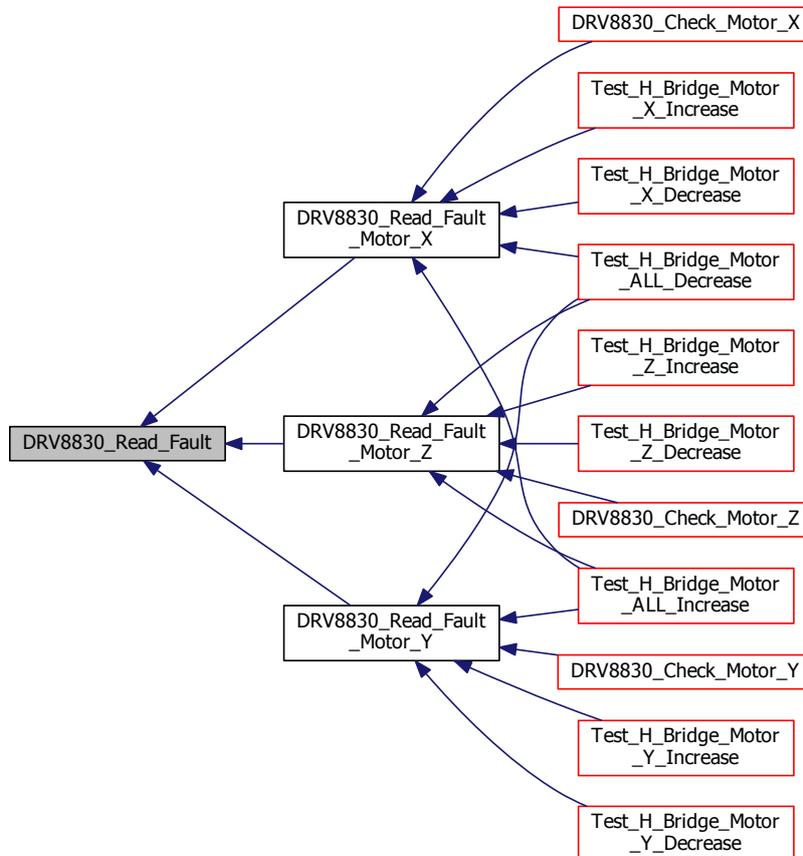
References [DRV8830_Fault](#), and [I2C_Interface_Single_Read\(\)](#).

Referenced by [DRV8830_Read_Fault_Motor_X\(\)](#), [DRV8830_Read_Fault_Motor_Y\(\)](#), and [DRV8830_Read_Fault_Motor_Z\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.50.4.13 uint8_t DRV8830_Read_Fault_Motor_X(void)

This function reads the fault byte for the X-axis h-bridge.

Returns

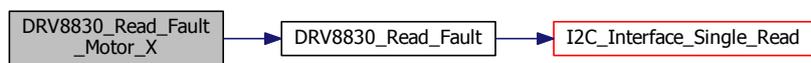
The current fault byte for the X-axis h-bridge.

Definition at line 512 of file [DRV8830.c](#).

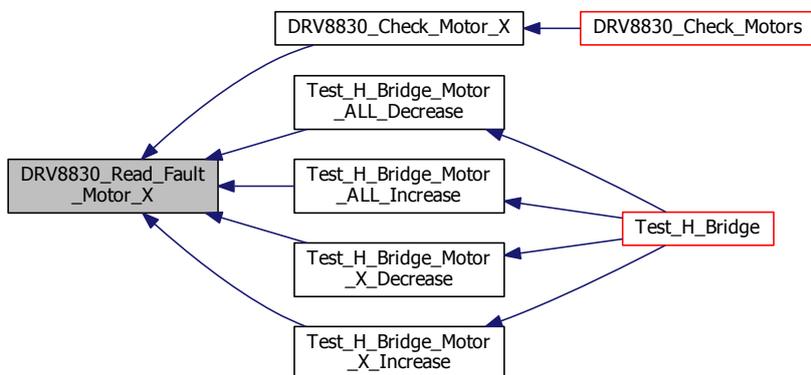
References [DRV8830_Read_Fault\(\)](#), and [Motor_Address_X](#).

Referenced by [DRV8830_Check_Motor_X\(\)](#), [Test_H_Bridge_Motor_ALL_Decrease\(\)](#), [Test_H_Bridge_Motor_ALL_Increase\(\)](#), [Test_H_Bridge_Motor_X_Decrease\(\)](#), and [Test_H_Bridge_Motor_X_Increase\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.50.4.14 uint8_t DRV8830_Read_Fault_Motor_Y(void)

This function reads the fault byte for the Y-axis h-bridge.

Returns

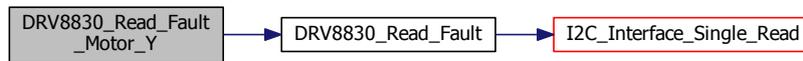
The current fault byte for the Y-axis h-bridge.

Definition at line 522 of file [DRV8830.c](#).

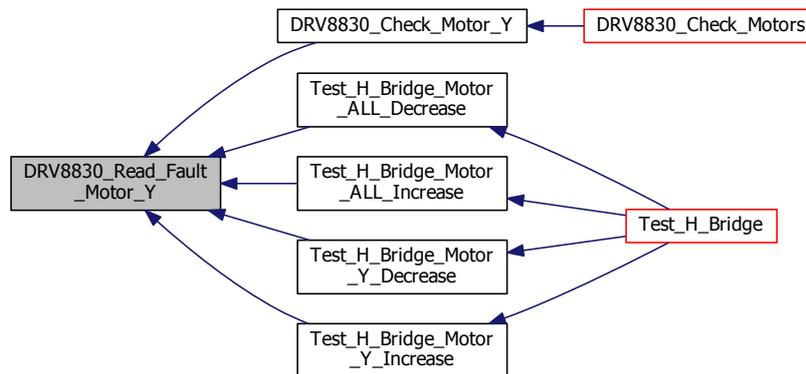
References [DRV8830_Read_Fault\(\)](#), and [Motor_Address_Y](#).

Referenced by [DRV8830_Check_Motor_Y\(\)](#), [Test_H_Bridge_Motor_ALL_Decrease\(\)](#), [Test_H_Bridge_Motor_ALL_Increase\(\)](#), [Test_H_Bridge_Motor_Y_Decrease\(\)](#), and [Test_H_Bridge_Motor_Y_Increase\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.50.4.15 uint8_t DRV8830_Read_Fault_Motor_Z (void)

This function reads the fault byte for the Z-axis h-bridge.

Returns

The current fault byte for the Z-axis h-bridge.

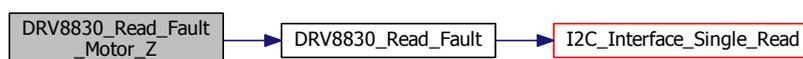
Return the fault byte for the Z-axis motor.

Definition at line 532 of file [DRV8830.c](#).

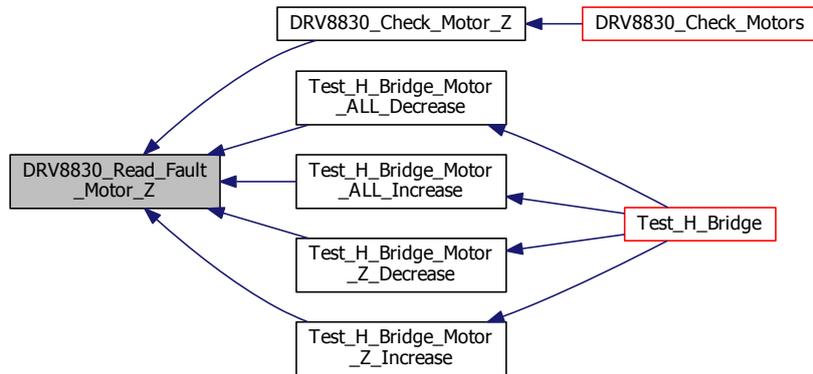
References [DRV8830_Read_Fault\(\)](#), and [Motor_Address_Z](#).

Referenced by [DRV8830_Check_Motor_Z\(\)](#), [Test_H_Bridge_Motor_ALL_Decrease\(\)](#), [Test_H_Bridge_Motor_ALL_Increase\(\)](#), [Test_H_Bridge_Motor_Z_Decrease\(\)](#), and [Test_H_Bridge_Motor_Z_Increase\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



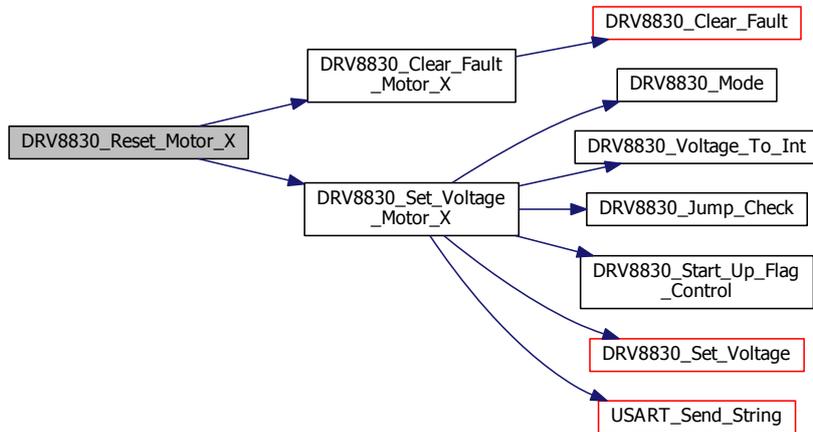
3.50.4.16 void DRV8830_Reset_Motor_X (void)

Definition at line 576 of file DRV8830.c.

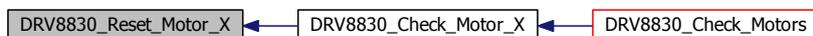
References DRV8830_Clear_Fault_Motor_X(), and DRV8830_Set_Voltage_Motor_X().

Referenced by DRV8830_Check_Motor_X().

Here is the call graph for this function:



Here is the caller graph for this function:



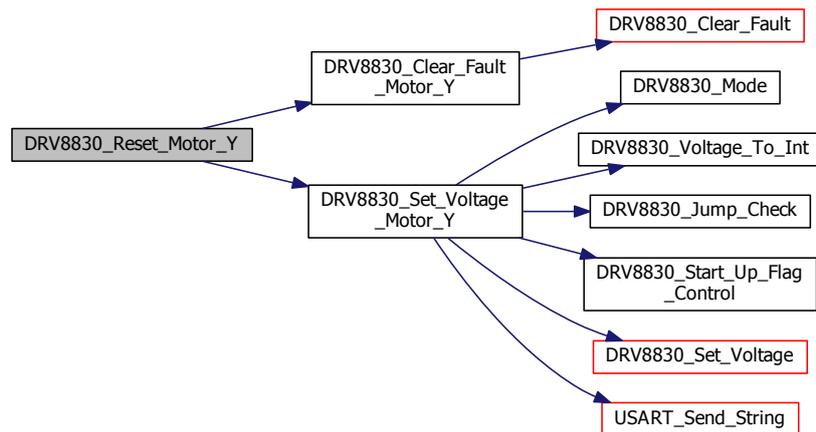
3.50.4.17 void DRV8830_Reset_Motor_Y (void)

Definition at line 584 of file [DRV8830.c](#).

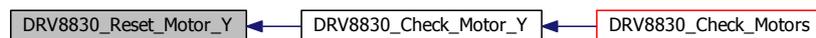
References [DRV8830_Clear_Fault_Motor_Y\(\)](#), and [DRV8830_Set_Voltage_Motor_Y\(\)](#).

Referenced by [DRV8830_Check_Motor_Y\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



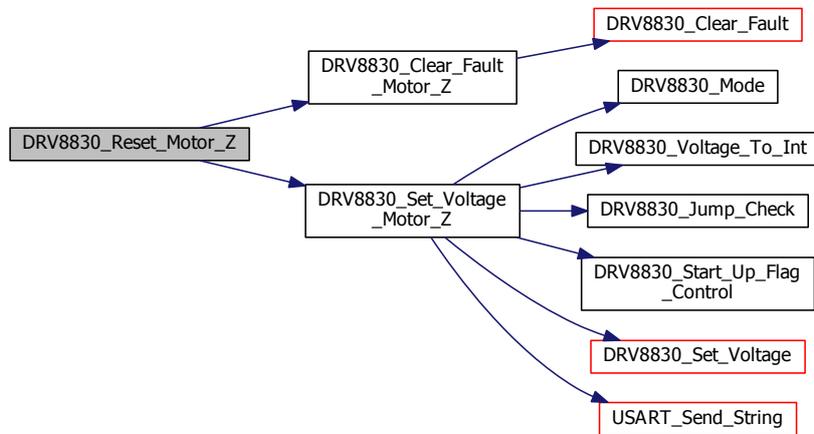
3.50.4.18 void DRV8830_Reset_Motor_Z (void)

Definition at line 592 of file [DRV8830.c](#).

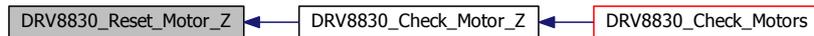
References [DRV8830_Clear_Fault_Motor_Z\(\)](#), and [DRV8830_Set_Voltage_Motor_Z\(\)](#).

Referenced by [DRV8830_Check_Motor_Z\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.50.4.19 void DRV8830_Set_Voltage (uint8_t Address, uint8_t Voltage_Value, uint8_t Mode) [inline]

This function sets the output voltage for the h-bridge.

Parameters

<i>Address</i>	The h-bridge I2C address.
<i>Voltage_Value</i>	The equivalent voltage value for the h-bridge. Note this is not the desired voltage.
<i>Mode</i>	The direction for the h-bridge (0 Standby/coast, 1 Reverse, 2 Forward, 3 Brake).

Warning

This function does not check if the mode is above 3 (the maximum for the specified selection).

Definition at line 373 of file [DRV8830.c](#).

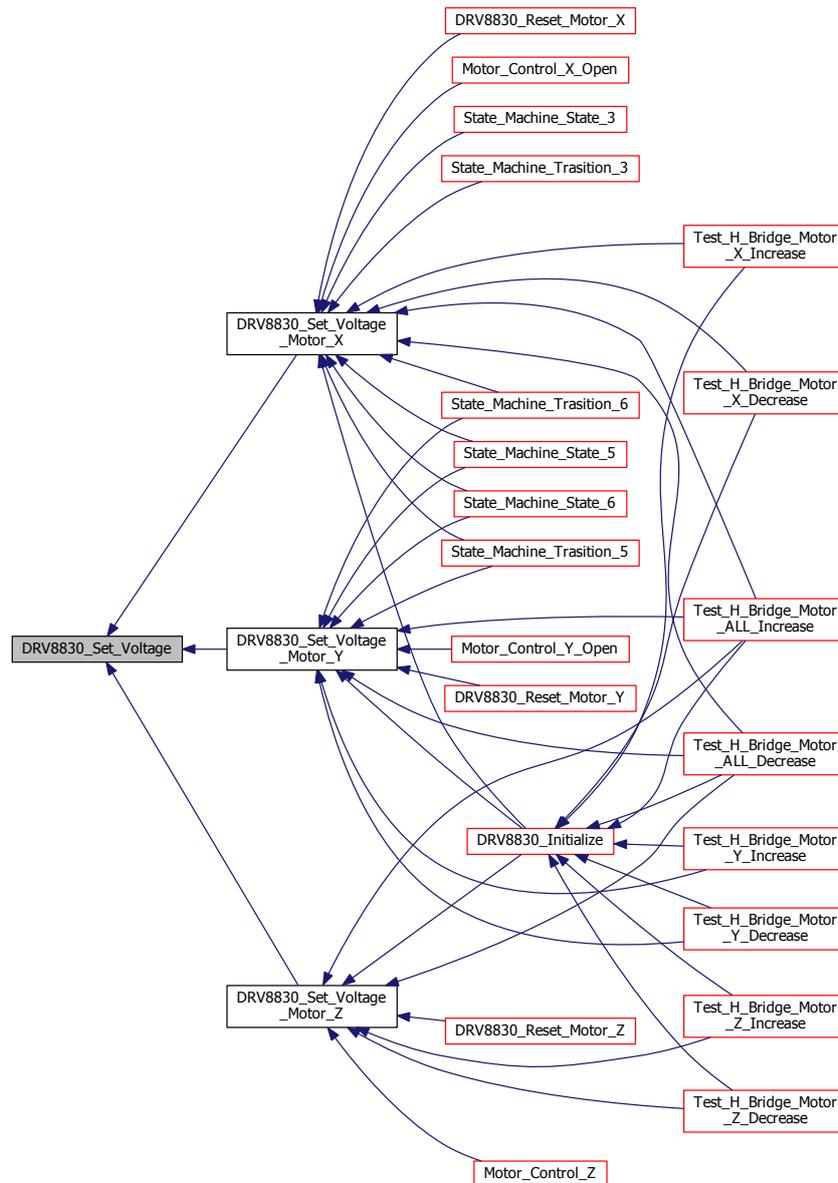
References [DRV8830_Control](#), and [I2C_Interface_Write\(\)](#).

Referenced by [DRV8830_Set_Voltage_Motor_X\(\)](#), [DRV8830_Set_Voltage_Motor_Y\(\)](#), and [DRV8830_Set_Voltage_Motor_Z\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.50.4.20 void DRV8830_Set_Voltage_Motor_X (float Voltage)

This function sets the output voltage for the X-axis h-bridge. Note that this function only Updates the h-bridge if the mode or voltage is different.

Parameters

<i>Voltage</i>	The desired voltage for the motor to be at.
----------------	---

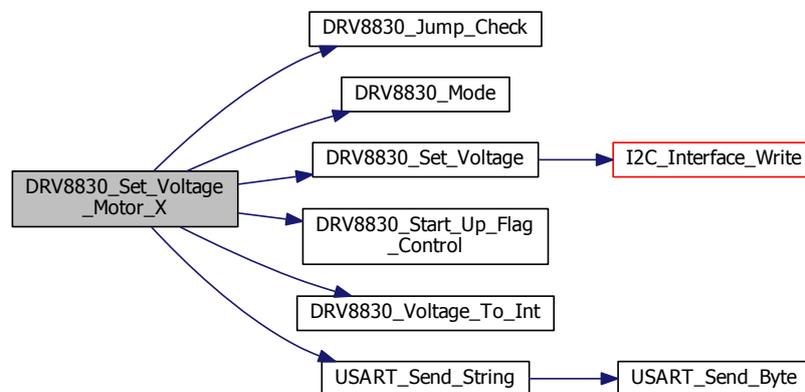
Definition at line 383 of file [DRV8830.c](#).

References [DRV8830_Jump_Check\(\)](#), [DRV8830_Mode\(\)](#), [DRV8830_Set_Voltage\(\)](#), [DRV8830_Start_Up_Flag_↔Control\(\)](#), [DRV8830_Voltage_To_Int\(\)](#), [Motor_Address_X](#), and [USART_Send_String\(\)](#).

Referenced by [DRV8830_Initialize\(\)](#), [DRV8830_Reset_Motor_X\(\)](#), [Motor_Control_X_Open\(\)](#), [State_Machine_↔](#)

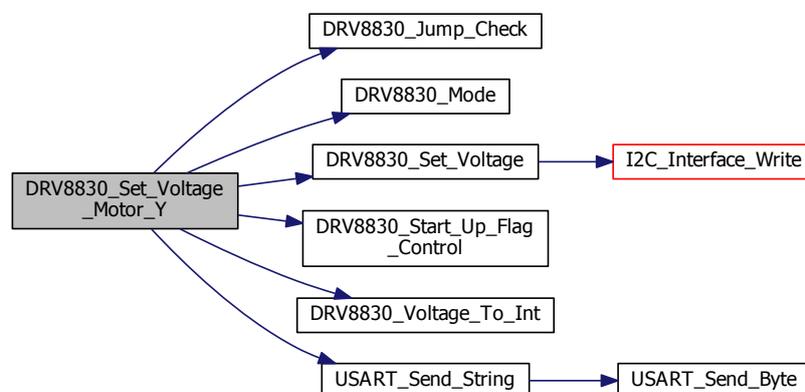
State_3(), State_Machine_State_5(), State_Machine_State_6(), State_Machine_Trasition_3(), State_Machine_↔
Trasition_5(), State_Machine_Trasition_6(), Test_H_Bridge_Motor_ALL_Decrease(), Test_H_Bridge_Motor_ALL_↔
_Increase(), Test_H_Bridge_Motor_X_Decrease(), and Test_H_Bridge_Motor_X_Increase().

Here is the call graph for this function:

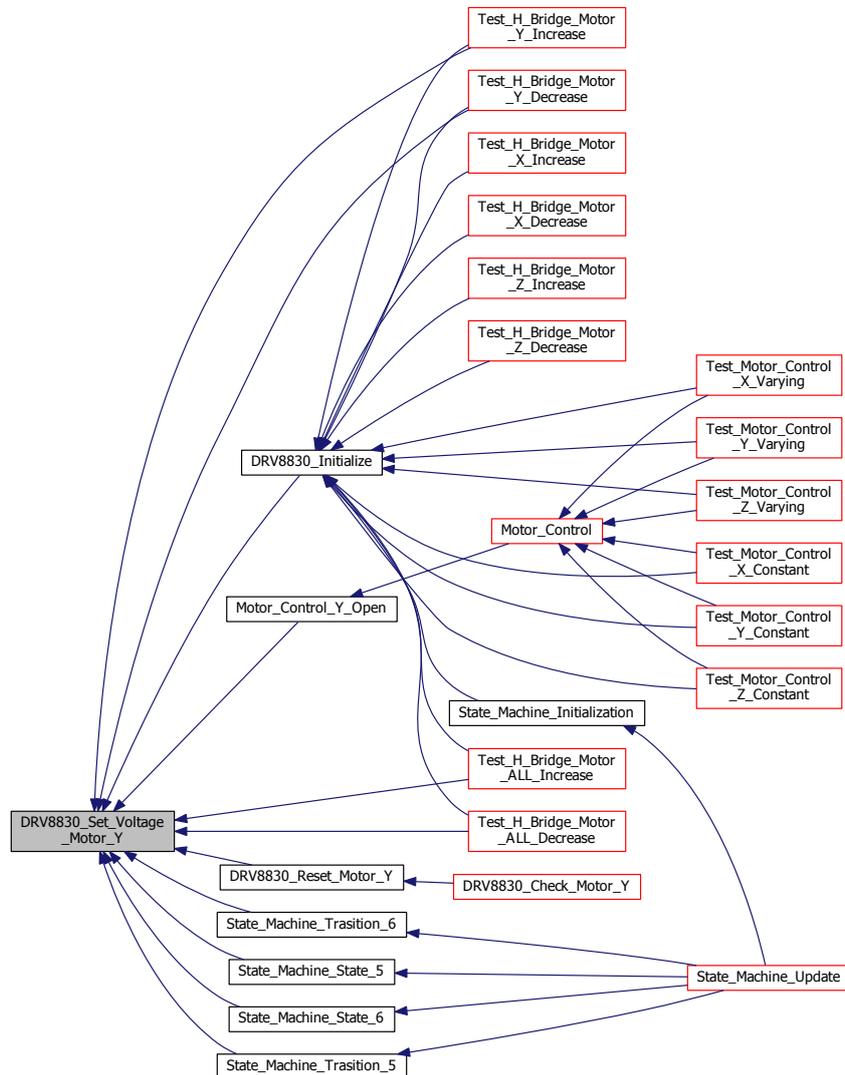


State_5(), State_Machine_State_6(), State_Machine_Trasition_5(), State_Machine_Trasition_6(), Test_H_Bridge_Motor_ALL_Decrease(), Test_H_Bridge_Motor_ALL_Increase(), Test_H_Bridge_Motor_Y_Decrease(), and Test_H_Bridge_Motor_Y_Increase().

Here is the call graph for this function:



Here is the caller graph for this function:



3.50.4.22 void DRV8830_Set_Voltage_Motor_Z (float Voltage)

This function sets the output voltage for the Z-axis h-bridge. Note that this function only Updates the h-bridge if the mode or voltage is different.

Parameters

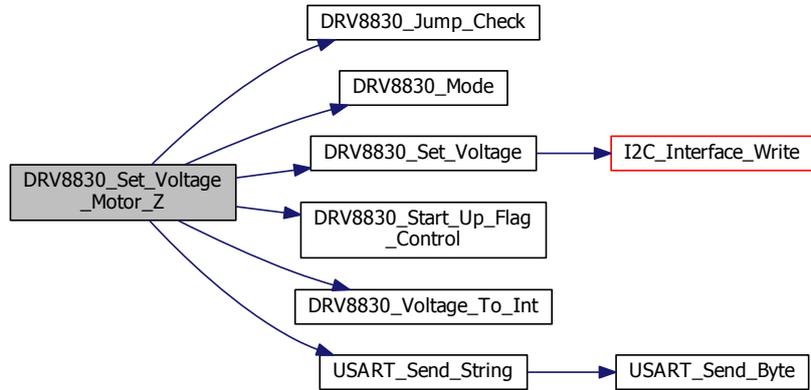
<i>Voltage</i>	The desired voltage for the motor to be at.
----------------	---

Definition at line 453 of file [DRV8830.c](#).

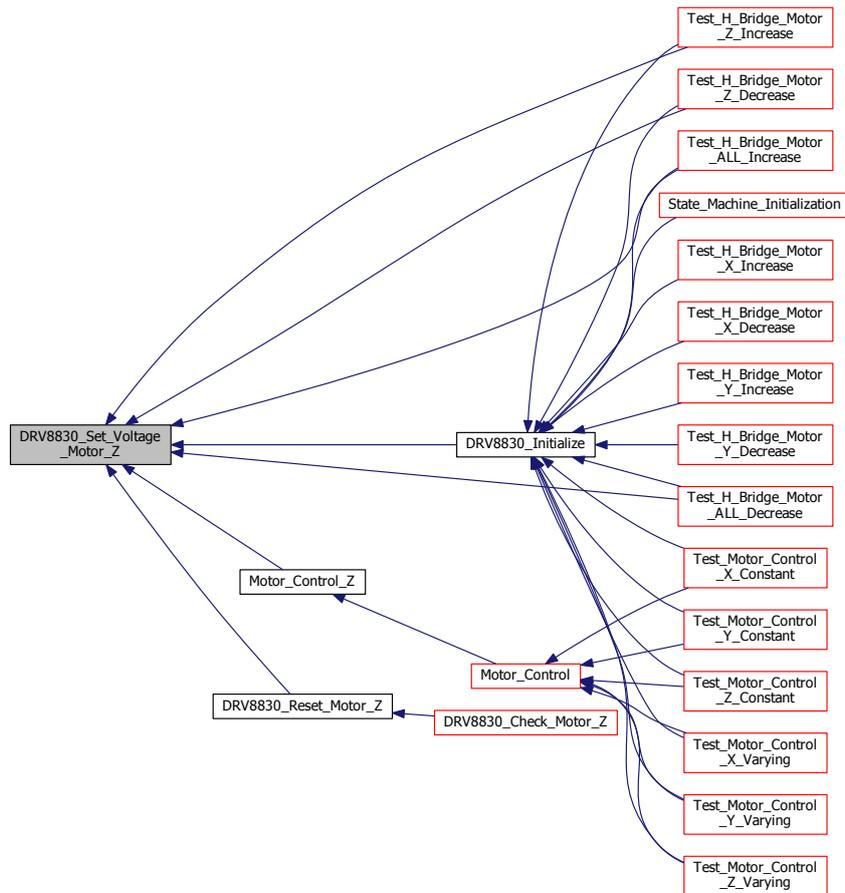
References [DRV8830_Jump_Check\(\)](#), [DRV8830_Mode\(\)](#), [DRV8830_Set_Voltage\(\)](#), [DRV8830_Start_Up_Flag_↔Control\(\)](#), [DRV8830_Voltage_To_Int\(\)](#), [Motor_Address_Z](#), and [USART_Send_String\(\)](#).

Referenced by [DRV8830_Initialize\(\)](#), [DRV8830_Reset_Motor_Z\(\)](#), [Motor_Control_Z\(\)](#), [Test_H_Bridge_Motor_AL↔L_Decrease\(\)](#), [Test_H_Bridge_Motor_ALL_Increase\(\)](#), [Test_H_Bridge_Motor_Z_Decrease\(\)](#), and [Test_H_Bridge↔_Motor_Z_Increase\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.50.4.23 uint8_t DRV8830_Start_Up_Flag_Control (uint8_t *Motor_Address*)

This function checks whether or not a motor should be updated. Making sure that there are no motors that are turning on at the same time.

Parameters

<i>Motor_Address</i>	The motor address of the motor to be updated.
----------------------	---

Returns

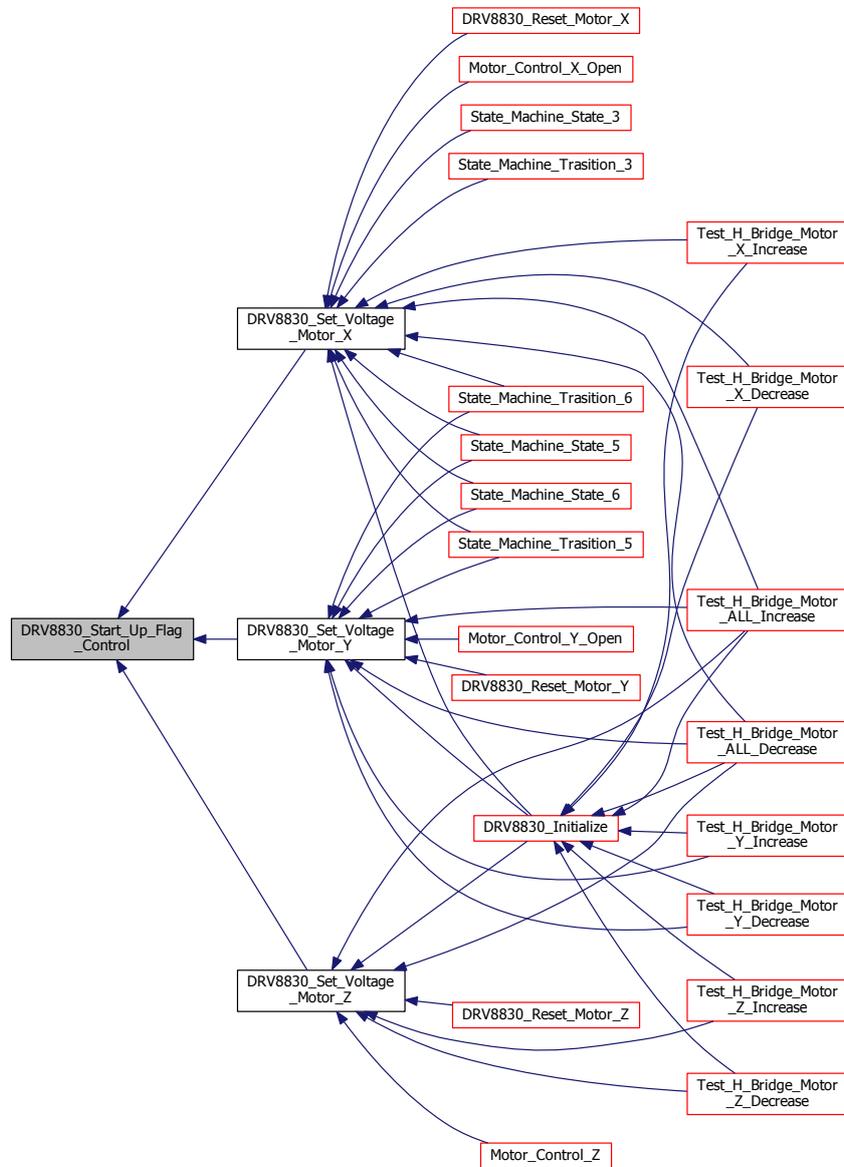
Whether or not the motor should update.

Definition at line 269 of file [DRV8830.c](#).

References [Motor_Address_X](#), [Motor_Address_Y](#), and [Motor_Address_Z](#).

Referenced by [DRV8830_Set_Voltage_Motor_X\(\)](#), [DRV8830_Set_Voltage_Motor_Y\(\)](#), and [DRV8830_Set_Voltage_Motor_Z\(\)](#).

Here is the caller graph for this function:



3.50.4.24 `uint8_t DRV8830_Voltage_To_Int (float Voltage) [inline]`

This function converts a desired voltage to the equivalent voltage value for the h-bridge.

Parameters

<i>Voltage</i>	The desired voltage for the motor to be at.
----------------	---

Returns

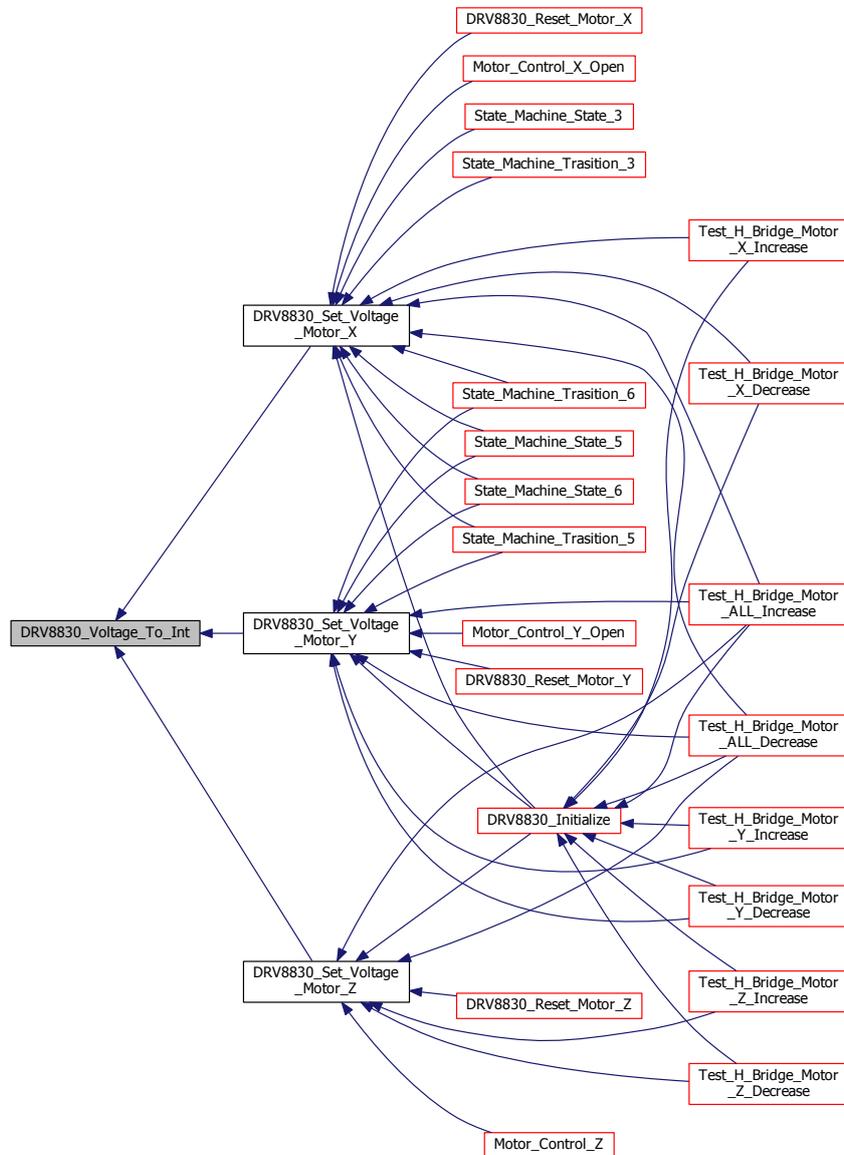
The equivalent voltage value for the h-bridge.

Definition at line 75 of file DRV8830.c.

References [H_Bridge_Max_Voltage_VSET](#), [H_Bridge_Min_Voltage](#), [H_Bridge_Voltage_Conversion](#), and [Motor_Max_Voltage](#).

Referenced by [DRV8830_Set_Voltage_Motor_X\(\)](#), [DRV8830_Set_Voltage_Motor_Y\(\)](#), and [DRV8830_Set_Voltage_Motor_Z\(\)](#).

Here is the caller graph for this function:



3.51 DRV8830.c

```

00001 /**
00002  * @file DRV8830.c
00003  * @author Nicholas Sikkema
00004  * @version Revision: 2.0
00005  * @date Last Updated: 4/21/2015\n Created: 11/8/2014 3:54:57 PM
00006  * @brief This file is code to read control the h-bridges.
00007  */
00008
00009 /** @brief The conversion factor used between the floating point value and the value for the h-bridge. */
00010 #define H_Bridge_Voltage_Conversion 12.5
00011 /** @brief The h-bridge value that is equivalent to Motor_Max_Voltage. */
00012 #define H_Bridge_Max_Voltage_VSET (uint8_t)(H_Bridge_Voltage_Conversion*Motor_Max_Voltage)
00013 /** @brief The minimum voltage that the h-bridge will allow. */
00014 #define H_Bridge_Min_Voltage 0.48
00015 /** @brief The h-bridge value that is equivalent to Motor_Max_Voltage. */
00016 #define H_Bridge_Min_Voltage_VSET (uint8_t)(H_Bridge_Voltage_Conversion*H_Bridge_Min_Voltage)
00017 /** @brief The clear fault bit for the h-bridge. */
00018 #define Clear_Fault_Bit 7
00019 /** @brief Maximum amount of times allowed to fault before sending that the motor failed. */
00020 #define Motor_Fault_Count_Max 5
00021 /** @brief Maximum voltage jump allowed for a motor. */
00022 #define Motor_Max_Jump 25
00023
00024 /** @brief I2C h-bridge addresses for the motors. */
00025 enum Motor_Adress
00026 {
00027     Motor_Address_X = 0x60,
00028     Motor_Address_Y = 0x62,
00029     Motor_Address_Z = 0x66
00030 };
00031
00032 /** @brief The register addresses for the DRV8830. */
00033 enum DRV8830_Registers
00034 {
00035     DRV8830_Control = 0x00,
00036     DRV8830_Fault = 0x01,
00037 };
00038
00039 /** @brief The list of motor modes for the DRV8830. */
00040 enum DRV8830_Modes
00041 {
00042     DRV8830_Standby = 0,
00043     DRV8830_Reverse = 1,
00044     DRV8830_Forward = 2,
00045     DRV8830_Stop = 3,
00046 };
00047
00048 #include <stdio.h>
00049 #include <avr/io.h>
00050 #include "../Master_Build_Control.h"
00051 #include "../Microcontroller/Communication/I2C_Interface.h"
00052 #include "../Microcontroller/Communication/USART.h"
00053 #include "DRV8830.h"
00054
00055 /* Static variables to keep track of the last voltage that was sent to the h-bridge. */
00056 static uint8_t Voltage_Value_X_Old = 0;
00057 static uint8_t Voltage_Value_Y_Old = 0;
00058 static uint8_t Voltage_Value_Z_Old = 0;
00059
00060 /* Static variables to keep track of the last mode that was sent to the h-bridge. */
00061 static uint8_t Mode_X_Old = 0;
00062 static uint8_t Mode_Y_Old = 0;
00063 static uint8_t Mode_Z_Old = 0;
00064
00065 /* Static variables to keep track of the motors that are trying to start-up. */
00066 static uint8_t Motor_X_Start_Up_Flag = 0;
00067 static uint8_t Motor_Y_Start_Up_Flag = 0;
00068 static uint8_t Motor_Z_Start_Up_Flag = 0;
00069
00070 /**
00071  * @brief This function converts a desired voltage to the equivalent voltage value for the h-bridge.
00072  * @param Voltage The desired voltage for the motor to be at.
00073  * @return The equivalent voltage value for the h-bridge.
00074  */
00075 inline uint8_t DRV8830_Voltage_To_Int(float Voltage)
00076 {
00077     /* Check to see if the input voltage is greater than the max rating of the motors that we are using. */
00078     if(Voltage>Motor_Max_Voltage)
00079     {
00080         /* To avoid burning up the motor if the input voltage is above the max voltage then return */
00081         return(H_Bridge_Max_Voltage_VSET);
00082     }
00083     /* Check to see if the input voltage is less than minimum voltage for the h-bridge. */
00084     else if(Voltage<H_Bridge_Min_Voltage)

```

```

00085     {
00086         /* If the input voltage is below the h-bridge minimum voltage, the only thing available is zero. */
00087         return(0);
00088     }
00089     /* Convert the voltage to a usable 8 bit value. */
00090     else
00091     {
00092         /* Return the converted h-bridge voltage value. */
00093         return((uint8_t)(Voltage*H_Bridge_Voltage_Conversion));
00094     }
00095 }
00096
00097 /**
00098  * @brief This function checks the desired equivalent voltage value.
00099  * Checking against the previous mode and voltage, to make sure of three conditions.
00100  * First is that the new voltage is not significantly greater than the previous voltage.
00101  * Second is to set the flags for the start-up of the motor. This will be used to make sure that
00102  * not more one motor is being started up at the same time.
00103  * Third is to prevent the motors from switching polarity too quickly.
00104  * @param New_Voltage The desired equivalent voltage value of the h-bridge.
00105  * Note that the function updates this variable with a new voltage if it matches on of the three
00106  * conditions.
00107  * @param New_Mode The desired mode of the h-bridge.
00108  * Note that the function updates this variable with a new mode if it matches on of the three conditions.
00109  * @param Old_Voltage The current equivalent voltage value of the h-bridge.
00110  * @param Old_Mode The mode of the h-bridge.
00111  * @return The start-up flag information for the motor.
00112  */
00112 inline uint8_t DRV8830_Jump_Check(uint8_t *New_Voltage, uint8_t *New_Mode, uint8_t
Old_Voltage, uint8_t Old_Mode)
00113 {
00114     /* Initialize the variable that holds the start-up flag information. */
00115     uint8_t Return_Value = 0;
00116     /* Check the current mode of the h-bridge. */
00117     switch (Old_Mode)
00118     {
00119         /* The current mode of the h-bridge is reverse. */
00120         case DRV8830_Reverse:
00121             /* Check the desired mode of the h-bridge. */
00122             switch(*New_Mode)
00123             {
00124                 /* The desired mode is reverse. */
00125                 case DRV8830_Reverse:
00126                     /* Limit a positive voltage jump. */
00127                     if(*New_Voltage-Old_Voltage>Motor_Max_Jump)
00128                     {
00129                         /* Increase the voltage by the maximum jump. */
00130                         *New_Voltage = Old_Voltage+Motor_Max_Jump;
00131                     }
00132                     /* Limit a negative voltage jump. */
00133                     else if(Old_Voltage-*New_Voltage>Motor_Max_Jump)
00134                     {
00135                         /* Decrease the voltage by the maximum jump. */
00136                         *New_Voltage = Old_Voltage-Motor_Max_Jump;
00137                     }
00138                     break;
00139                 /* The desired mode is forward. */
00140                 case DRV8830_Forward:
00141                     /* Check to see if the motor voltage can be decreased without going to a complete stop.
00142                     */
00143                     if(Old_Voltage>=Motor_Max_Jump+
H_Bridge_Min_Voltage_VSET)
00144                     {
00145                         /* Decrease the motor voltage. */
00146                         *New_Voltage = Old_Voltage-Motor_Max_Jump;
00147                         *New_Mode = Old_Mode;
00148                     }
00149                     else
00150                     {
00151                         /* Stop the motor by setting the mode to stop and voltage to 0. */
00152                         *New_Voltage = 0;
00153                         *New_Mode = DRV8830_Stop;
00154                     }
00155                     break;
00156                 }
00157             }
00158             /* The current mode of the h-bridge is forward. */
00159             case DRV8830_Forward:
00160                 /* Check the desired mode of the h-bridge. */
00161                 switch(*New_Mode)
00162                 {
00163                     /* The desired mode is reverse. */
00164                     case DRV8830_Reverse:
00165                         /* Check to see if the motor voltage can be decreased without going to a complete stop.
00166                         */
00167                         if(Old_Voltage>=Motor_Max_Jump+
H_Bridge_Min_Voltage_VSET)

```

```

00166     {
00167         /* Decrease the motor voltage. */
00168         *New_Voltage = Old_Voltage-Motor_Max_Jump;
00169         *New_Mode = Old_Mode;
00170     }
00171     else
00172     {
00173         /* Stop the motor by setting the mode to stop and voltage to 0. */
00174         *New_Voltage = 0;
00175         *New_Mode = DRV8830_Stop;
00176     }
00177     break;
00178     /* The desired mode is forward. */
00179     case DRV8830_Forward:
00180         /* Limit a positive voltage jump. */
00181         if(*New_Voltage-Old_Voltage>Motor_Max_Jump)
00182         {
00183             /* Increase the voltage by the maximum jump. */
00184             *New_Voltage = Old_Voltage+Motor_Max_Jump;
00185         }
00186         /* Limit a negative voltage jump. */
00187         else if(Old_Voltage-*New_Voltage>Motor_Max_Jump)
00188         {
00189             /* Decrease the voltage by the maximum jump. */
00190             *New_Voltage = Old_Voltage-Motor_Max_Jump;
00191         }
00192         break;
00193     }
00194     break;
00195     /* The current mode of the h-bridge is stop. */
00196     default:
00197         /* Check the desired mode of the h-bridge. */
00198         switch(*New_Mode)
00199         {
00200             /* The desired mode is reverse. */
00201             case DRV8830_Reverse:
00202                 /* Set the start-up flag. */
00203                 Return_Value = 1;
00204                 /* Limit a positive voltage jump. */
00205                 if(*New_Voltage-Old_Voltage>Motor_Max_Jump)
00206                 {
00207                     /* Increase the voltage by the maximum jump. */
00208                     *New_Voltage = Old_Voltage+Motor_Max_Jump;
00209                 }
00210                 break;
00211             /* The desired mode is forward. */
00212             case DRV8830_Forward:
00213                 /* Set the start-up flag. */
00214                 Return_Value = 1;
00215                 /* Limit a positive voltage jump. */
00216                 if(*New_Voltage-Old_Voltage>Motor_Max_Jump)
00217                 {
00218                     /* Increase the voltage by the maximum jump. */
00219                     *New_Voltage = Old_Voltage+Motor_Max_Jump;
00220                 }
00221                 break;
00222         }
00223     }
00224     break;
00225     /* Return the start-up flag information. */
00226     return(Return_Value);
00227 }
00228
00229 /**
00230  * @brief This function uses the a desired voltage to determine the mode for the h-bridge.
00231  * @param Voltage The desired voltage for the motor to be at.
00232  * @return The new mode for the h-bridge.
00233  */
00234 inline uint8_t DRV8830_Mode(float *Voltage)
00235 {
00236     /* Initialize the variable that holds mode information. */
00237     uint8_t Return_Value = DRV8830_Stop;
00238     /* Check to see if the voltage is positive or negative. */
00239     if(*Voltage<0.0)
00240     {
00241         /* Invert the voltage if the voltage is negative. */
00242         *Voltage*=-1;
00243         /* Check to see if the voltage is above or equal to the minimum voltage. */
00244         if(*Voltage>=H_Bridge_Min_Voltage)
00245         {
00246             /* Set the motor mode to reverse. */
00247             Return_Value = DRV8830_Reverse;
00248         }
00249     }
00250     else
00251     {
00252         /* Check to see if the voltage is above or equal to the minimum voltage. */

```

```

00253         if(*Voltage>=H_Bridge_Min_Voltage)
00254         {
00255             /* Set the motor mode to forward. */
00256             Return_Value = DRV8830_Forward;
00257         }
00258     }
00259     /* Return the mode information. */
00260     return(Return_Value);
00261 }
00262
00263 /**
00264  * @brief This function checks whether or not a motor should be updated.
00265  * Making sure that there are no motors that are turning on at the same time.
00266  * @param Motor_Address The motor address of the motor to be updated.
00267  * @return Whether or not the motor should update.
00268  */
00269 uint8_t DRV8830_Start_Up_Flag_Control(uint8_t Motor_Address)
00270 {
00271     /* Initialize the variable that holds the information needed for updating the motor. */
00272     uint8_t Return_Value = 0;
00273     /* Variables to keep track of which motor is starting up. */
00274     static uint8_t Start_Up_X = 0;
00275     static uint8_t Start_Up_Y = 0;
00276     static uint8_t Start_Up_Z = 0;
00277     /* Check the motor address. */
00278     switch(Motor_Address)
00279     {
00280         /* Check the information for the X motor. */
00281         case Motor_Address_X:
00282             /* Check if the routine wants to start-up the motor. */
00283             if(Motor_X_Start_Up_Flag)
00284             {
00285                 /* Check if another motor is starting up. */
00286                 if(Start_Up_Y||Start_Up_Z)
00287                 {
00288                     /* Keep the motor off due to another motor starting. */
00289                     Start_Up_X = 0;
00290                 }
00291                 else
00292                 {
00293                     /* Allow the motor to update. */
00294                     Return_Value = 1;
00295                     /* Start-up the motor. */
00296                     Start_Up_X = 1;
00297                 }
00298             }
00299         else
00300         {
00301             /* Normal operation of the motor. */
00302             Return_Value = 1;
00303             /* Clear the startup flag. */
00304             Start_Up_X = 0;
00305         }
00306         break;
00307         /* Check the information for the Y motor. */
00308         case Motor_Address_Y:
00309             /* Check if the routine wants to start-up the motor. */
00310             if(Motor_Y_Start_Up_Flag)
00311             {
00312                 /* Check if another motor is starting up. */
00313                 if(Start_Up_X||Start_Up_Z)
00314                 {
00315                     /* Keep the motor off due to another motor starting and clear the start-up variable. */
00316                     Start_Up_Y = 0;
00317                 }
00318                 else
00319                 {
00320                     /* Allow the motor to update. */
00321                     Return_Value = 1;
00322                     /* Start-up the motor. */
00323                     Start_Up_Y = 1;
00324                 }
00325             }
00326         else
00327         {
00328             /* Normal operation of the motor. */
00329             Return_Value = 1;
00330             /* Clear the startup flag. */
00331             Start_Up_Y = 0;
00332         }
00333         break;
00334         /* Check the information for the Z motor. */
00335         case Motor_Address_Z:
00336             /* Check if the routine wants to start-up the motor. */
00337             if(Motor_Z_Start_Up_Flag)
00338             {
00339                 /* Check if another motor is starting up. */

```

```

00340         if(Start_Up_X||Start_Up_Y)
00341         {
00342             /* Keep the motor off due to another motor starting and clear the start-up variable. */
00343             Start_Up_Z = 0;
00344         }
00345         else
00346         {
00347             /* Allow the motor to update. */
00348             Return_Value = 1;
00349             /* Start-up the motor. */
00350             Start_Up_Z = 1;
00351         }
00352     }
00353     else
00354     {
00355         /* Normal operation of the motor. */
00356         Return_Value = 1;
00357         /* Clear the startup flag. */
00358         Start_Up_Z = 0;
00359     }
00360     break;
00361 }
00362 /* Return the information needed for updating the motor. */
00363 return(Return_Value);
00364 }
00365
00366 /**
00367  * @brief This function sets the output voltage for the h-bridge.
00368  * @param Address The h-bridge I2C address.
00369  * @param Voltage_Value The equivalent voltage value for the h-bridge. Note this is not the desired
00370  * voltage.
00371  * @param Mode The direction for the h-bridge (0 Standby/coast, 1 Reverse, 2 Forward, 3 Brake).
00372  * @warning This function does not check if the mode is above 3 (the maximum for the specified selection).
00373  */
00373 inline void DRV8830_Set_Voltage(uint8_t Address, uint8_t Voltage_Value, uint8_t Mode)
00374 {
00375     /* Send the update voltage message to the h-bridge. */
00376     I2C_Interface_Write(Address, DRV8830_Control, (Voltage_Value<<2)|Mode
00377 );
00378 }
00379 /**
00380  * @brief This function sets the output voltage for the X-axis h-bridge. Note that this function only
00381  * updates the h-bridge if the mode or voltage is different.
00382  * @param Voltage The desired voltage for the motor to be at.
00383  */
00383 void DRV8830_Set_Voltage_Motor_X(float Voltage)
00384 {
00385     uint8_t Mode = DRV8830_Mode(&Voltage);
00386     /* Convert the voltage value to a h-bridge value.*/
00387     uint8_t Voltage_Value = DRV8830_Voltage_To_Int(Voltage);
00388     /* Check to if the voltage value and the mode are the same as the previous values. */
00389     if (Voltage_Value_X_Old != Voltage_Value || Mode_X_Old != Mode)
00390     {
00391         /* Check the start up flag using the new values for the X motors. */
00392         Motor_X_Start_Up_Flag = DRV8830_Jump_Check(&Voltage_Value, &Mode,
00393 Voltage_Value_X_Old, Mode_X_Old);
00394         /* Check the check the start up flags. */
00394         if(DRV8830_Start_Up_Flag_Control(
00395 Motor_Address_X))
00396         {
00396             /* Update the h-bridge voltage value for the X-axis motor. */
00397             DRV8830_Set_Voltage(Motor_Address_X, Voltage_Value, Mode);
00398             /* Update the previous voltage value for the X-axis motor. */
00399             Voltage_Value_X_Old = Voltage_Value;
00400             /* Update the previous mode for the X-axis motor. */
00401             Mode_X_Old = Mode;
00402         }
00403         #if H_Bridge_Print == 1
00404             /* Initialize a character array. */
00405             char Buffer[50];
00406             /* Setup the buffer to send. */
00407             sprintf(Buffer, "X V %u, M %u, F %u\r\n", Voltage_Value_X_Old, Mode_X_Old,
00408 Motor_X_Start_Up_Flag);
00408             /* Send the string over USART. */
00409             USART_Send_String(Buffer);
00410         #endif
00411     }
00412 }
00413
00414 /**
00415  * @brief This function sets the output voltage for the Y-axis h-bridge. Note that this function only
00416  * updates the h-bridge if the mode or voltage is different.
00417  * @param Voltage The desired voltage for the motor to be at.
00418  */
00418 void DRV8830_Set_Voltage_Motor_Y(float Voltage)
00419 {

```

```

00420     uint8_t Mode = DRV8830_Mode(&Voltage);
00421     /* Convert the voltage value to a h-bridge value.*/
00422     uint8_t Voltage_Value = DRV8830_Voltage_To_Int(Voltage);
00423     /* Check to if the voltage value and the mode are the same as the previous values. */
00424     if (Voltage_Value_Y_Old != Voltage_Value || Mode_Y_Old != Mode)
00425     {
00426         /* Check the start up flag using the new values for the Y motors. */
00427         Motor_Y_Start_Up_Flag = DRV8830_Jump_Check(&Voltage_Value, &Mode,
Voltage_Value_Y_Old, Mode_Y_Old);
00428         /* Check the check the start up flags. */
00429         if(DRV8830_Start_Up_Flag_Control(
Motor_Address_Y))
00430         {
00431             /* Update the h-bridge voltage value for the X-axis motor. */
00432             DRV8830_Set_Voltage(Motor_Address_Y, Voltage_Value, Mode);
00433             /* Update the previous voltage value for the X-axis motor. */
00434             Voltage_Value_Y_Old = Voltage_Value;
00435             /* Update the previous mode for the X-axis motor. */
00436             Mode_Y_Old = Mode;
00437         }
00438         #if H_Bridge_Print == 1
00439             /* Initialize a character array. */
00440             char Buffer[50];
00441             /* Setup the buffer to send. */
00442             sprintf(Buffer, "Y V %u, M %u, F %u\r\n", Voltage_Value_Y_Old, Mode_Y_Old,
Motor_Y_Start_Up_Flag);
00443             /* Send the string over USART. */
00444             USART_Send_String(Buffer);
00445             #endif
00446         }
00447     }
00448
00449 /**
00450  * @brief This function sets the output voltage for the Z-axis h-bridge. Note that this function only
Updates the h-bridge if the mode or voltage is different.
00451  * @param Voltage The desired voltage for the motor to be at.
00452  */
00453 void DRV8830_Set_Voltage_Motor_Z(float Voltage)
00454 {
00455     uint8_t Mode = DRV8830_Mode(&Voltage);
00456     /* Convert the voltage value to a h-bridge value.*/
00457     uint8_t Voltage_Value = DRV8830_Voltage_To_Int(Voltage);
00458     /* Check to if the voltage value and the mode are the same as the previous values. */
00459     if (Voltage_Value_Z_Old != Voltage_Value || Mode_Z_Old != Mode)
00460     {
00461         /* Check the start up flag using the new values for the Z motors. */
00462         Motor_Z_Start_Up_Flag = DRV8830_Jump_Check(&Voltage_Value, &Mode,
Voltage_Value_Z_Old, Mode_Z_Old);
00463         /* Check the check the start up flags. */
00464         if(DRV8830_Start_Up_Flag_Control(
Motor_Address_Z))
00465         {
00466             /* Update the h-bridge voltage value for the X-axis motor. */
00467             DRV8830_Set_Voltage(Motor_Address_Z, Voltage_Value, Mode);
00468             /* Update the previous voltage value for the X-axis motor. */
00469             Voltage_Value_Z_Old = Voltage_Value;
00470             /* Update the previous mode for the X-axis motor. */
00471             Mode_Z_Old = Mode;
00472         }
00473         #if H_Bridge_Print == 1
00474             /* Initialize a character array. */
00475             char Buffer[50];
00476             /* Setup the buffer to send. */
00477             sprintf(Buffer, "Z V %u, M %u, F %u\r\n", Voltage_Value_Z_Old, Mode_Z_Old,
Motor_Z_Start_Up_Flag);
00478             /* Send the string over USART. */
00479             USART_Send_String(Buffer);
00480             #endif
00481         }
00482     }
00483
00484 /**
00485  * @brief This function sets the initial output voltage for all three h-bridges.
00486  */
00487 inline void DRV8830_Initialize(void)
00488 {
00489     /* Initialize the X-axis motor to 0 volts. */
00490     DRV8830_Set_Voltage_Motor_X(0.0);
00491     /* Initialize the Y-axis motor to 0 volts. */
00492     DRV8830_Set_Voltage_Motor_Y(0.0);
00493     /* Initialize the Z-axis motor to 0 volts. */
00494     DRV8830_Set_Voltage_Motor_Z(0.0);
00495 }
00496
00497 /**
00498  * @brief This function reads the fault byte for the h-bridge.
00499  * @param Address The h-bridge I2C address.

```

```

00500  * @return The current fault byte for the h-bridge.
00501  */
00502  inline uint8_t DRV8830_Read_Fault(uint8_t Address)
00503  {
00504      /* Read and send back the contents of the fault byte. */
00505      return(I2C_Interface_Single_Read(Address,
DRV8830_Fault));
00506  }
00507
00508  /**
00509  * @brief This function reads the fault byte for the X-axis h-bridge.
00510  * @return The current fault byte for the X-axis h-bridge.
00511  */
00512  uint8_t DRV8830_Read_Fault_Motor_X(void)
00513  {
00514      /* Return the fault byte for the X-axis motor. */
00515      return(DRV8830_Read_Fault(Motor_Address_X));
00516  }
00517
00518  /**
00519  * @brief This function reads the fault byte for the Y-axis h-bridge.
00520  * @return The current fault byte for the Y-axis h-bridge.
00521  */
00522  uint8_t DRV8830_Read_Fault_Motor_Y(void)
00523  {
00524      /* Return the fault byte for the Y-axis motor. */
00525      return(DRV8830_Read_Fault(Motor_Address_Y));
00526  }
00527
00528  /**
00529  * @brief This function reads the fault byte for the Z-axis h-bridge.
00530  * @return The current fault byte for the Z-axis h-bridge.
00531  */
00532  uint8_t DRV8830_Read_Fault_Motor_Z(void)
00533  {
00534      /** Return the fault byte for the Z-axis motor. */
00535      return(DRV8830_Read_Fault(Motor_Address_Z));
00536  }
00537
00538  /**
00539  * @brief This function clears the fault byte for the h-bridge.
00540  * @param Address The h-bridge I2C address.
00541  */
00542  inline void DRV8830_Clear_Fault(uint8_t Address)
00543  {
00544      /* Send the I2C message to the h-bridge to clear the fault bit. */
00545      I2C_Interface_Write(Address, DRV8830_Fault, 1<<
Clear_Fault_Bit);
00546  }
00547
00548  /**
00549  * @brief This function clears the fault byte for the X-axis h-bridge.
00550  */
00551  void DRV8830_Clear_Fault_Motor_X(void)
00552  {
00553      /* Clear the fault byte for the X-axis motor. */
00554      DRV8830_Clear_Fault(Motor_Address_X);
00555  }
00556
00557  /**
00558  * @brief This function clears the fault byte for the Y-axis h-bridge.
00559  */
00560  void DRV8830_Clear_Fault_Motor_Y(void)
00561  {
00562      /* Clear the fault byte for the Y-axis motor. */
00563      DRV8830_Clear_Fault(Motor_Address_Y);
00564  }
00565
00566  /**
00567  * @brief This function clears the fault byte for the Z-axis h-bridge.
00568  */
00569  void DRV8830_Clear_Fault_Motor_Z(void)
00570  {
00571      /* Clear the fault byte for the Z-axis motor. */
00572      DRV8830_Clear_Fault(Motor_Address_Z);
00573  }
00574
00575
00576  void DRV8830_Reset_Motor_X(void)
00577  {
00578      /* Clear fault previous fault values. */
00579      DRV8830_Clear_Fault_Motor_X();
00580      /* Set the motor voltage to 0. */
00581      DRV8830_Set_Voltage_Motor_X(0.0);
00582  }
00583
00584  void DRV8830_Reset_Motor_Y(void)

```

```

00585 {
00586     /* Clear fault previous fault values. */
00587     DRV8830_Clear_Fault_Motor_Y();
00588     /* Set the motor voltage to 0. */
00589     DRV8830_Set_Voltage_Motor_Y(0.0);
00590 }
00591
00592 void DRV8830_Reset_Motor_Z(void)
00593 {
00594     /* Clear fault previous fault values. */
00595     DRV8830_Clear_Fault_Motor_Z();
00596     /* Set the motor voltage to 0. */
00597     DRV8830_Set_Voltage_Motor_Z(0.0);
00598 }
00599
00600 /**
00601  * @brief This function checks to see if the X-axis motor is failing.
00602  * Counting and resetting the motor faults to make sure the motor is able to continue.
00603  * @return Whether or not the fault count is greater than a specified amount.
00604  */
00605 uint8_t DRV8830_Check_Motor_X(void)
00606 {
00607     /* Initialize the fault count for amount of motor failures. */
00608     static uint8_t Fault_Count_X = 0;
00609     /* Read the fault register from the X-axis motor h-bridge. */
00610     uint8_t Fault_Value = DRV8830_Read_Fault_Motor_X();
00611     if(Fault_Value&0x01)
00612     {
00613         /* Clear the fault register for the X-axis motor h-bridge. */
00614         DRV8830_Reset_Motor_X();
00615         /* Increment the fault count for the Y-axis motor. */
00616         Fault_Count_X++;
00617     }
00618     else if(Fault_Value)
00619     {
00620         /* Clear fault previous fault values. */
00621         DRV8830_Clear_Fault_Motor_X();
00622     }
00623     else
00624     {
00625         /* Clear the fault count for the X-axis motor. */
00626         Fault_Count_X = 0;
00627     }
00628     /* Return whether or not the fault count is greater than a specified amount. */
00629     return(Fault_Count_X>Motor_Fault_Count_Max);
00630 }
00631
00632 /**
00633  * @brief This function checks to see if the Y-axis motor is failing.
00634  * Counting and resetting the motor faults to make sure the motor is able to continue.
00635  * @return Whether or not the fault count is greater than a specified amount.
00636  */
00637 uint8_t DRV8830_Check_Motor_Y(void)
00638 {
00639     /* Initialize the fault count for amount of motor failures. */
00640     static uint8_t Fault_Count_Y = 0;
00641     /* Read the fault register from the Y-axis motor h-bridge. */
00642     uint8_t Fault_Value = DRV8830_Read_Fault_Motor_Y();
00643     if(Fault_Value&0x01)
00644     {
00645         /* Clear the fault register for the Y-axis motor h-bridge. */
00646         DRV8830_Reset_Motor_Y();
00647         /* Increment the fault count for the Y-axis motor. */
00648         Fault_Count_Y++;
00649     }
00650     else if(Fault_Value)
00651     {
00652         /* Clear fault previous fault values. */
00653         DRV8830_Clear_Fault_Motor_Y();
00654     }
00655     else
00656     {
00657         /* Clear the fault count for the Y-axis motor. */
00658         Fault_Count_Y = 0;
00659     }
00660     /* Return whether or not the fault count is greater than a specified amount. */
00661     return(Fault_Count_Y>Motor_Fault_Count_Max);
00662 }
00663
00664 /**
00665  * @brief This function checks to see if the Z-axis motor is failing.
00666  * Counting and resetting the motor faults to make sure the motor is able to continue.
00667  * @return Whether or not the fault count is greater than a specified amount.
00668  */
00669 uint8_t DRV8830_Check_Motor_Z(void)
00670 {
00671     /* Initialize the fault count for amount of motor failures. */

```

```

00672     static uint8_t Fault_Count_Z = 0;
00673     /* Read the fault register from the Z-axis motor h-bridge. */
00674     uint8_t Fault_Value = DRV8830_Read_Fault_Motor_Z();
00675     if(Fault_Value&0x01)
00676     {
00677         /* Clear the fault register for the Z-axis motor h-bridge. */
00678         DRV8830_Reset_Motor_Z();
00679         /* Increment the fault count for the Z-axis motor. */
00680         Fault_Count_Z++;
00681     }
00682     else if(Fault_Value)
00683     {
00684         /* Clear fault previous fault values. */
00685         DRV8830_Clear_Fault_Motor_Z();
00686     }
00687     else
00688     {
00689         /* Clear the fault count for the Z-axis motor. */
00690         Fault_Count_Z = 0;
00691     }
00692     /* Return whether or not the fault count is greater than a specified amount. */
00693     return(Fault_Count_Z>Motor_Fault_Count_Max);
00694 }
00695
00696 /**
00697  * @brief This function checks to see if any of the 3 motors is failing.
00698  * @return Whether or not the fault count is greater than a specified amount for each motor.\n
00699  * Bit 0: Z-axis motor\n
00700  * Bit 1: Y-axis motor\n
00701  * Bit 2: X-axis motor\n
00702  */
00703 uint8_t DRV8830_Check_Motors(void)
00704 {
00705     /* Return the value for the fault for each of the motors. Simplifying the return value into 3 bits.*/
00706     return(DRV8830_Check_Motor_X()<<2|DRV8830_Check_Motor_Y()<<1|
DRV8830_Check_Motor_Z()<<0);
00707 }

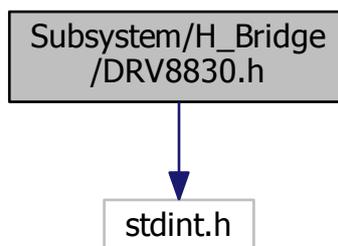
```

3.52 Subsystem/H_Bridge/DRV8830.h File Reference

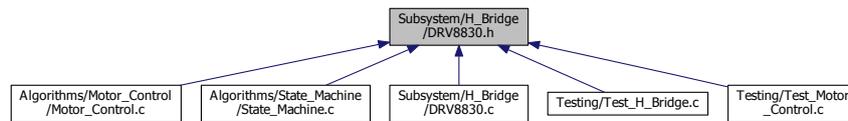
This file is code to read control the h-bridges.

```
#include <stdint.h>
```

Include dependency graph for DRV8830.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#define Motor_Max_Voltage 4.98`
The maximum voltage that we will allow for the h-bridge.

Functions

- `void DRV8830_Initialize (void)`
This function sets the initial output voltage for all three h-bridges.
- `void DRV8830_Set_Voltage_Motor_X (float Voltage)`
This function sets the output voltage for the X-axis h-bridge. Note that this function only Updates the h-bridge if the mode or voltage is different.
- `void DRV8830_Set_Voltage_Motor_Y (float Voltage)`
This function sets the output voltage for the Y-axis h-bridge. Note that this function only Updates the h-bridge if the mode or voltage is different.
- `void DRV8830_Set_Voltage_Motor_Z (float Voltage)`
This function sets the output voltage for the Z-axis h-bridge. Note that this function only Updates the h-bridge if the mode or voltage is different.
- `uint8_t DRV8830_Read_Fault_Motor_X (void)`
This function reads the fault byte for the X-axis h-bridge.
- `uint8_t DRV8830_Read_Fault_Motor_Y (void)`
This function reads the fault byte for the Y-axis h-bridge.
- `uint8_t DRV8830_Read_Fault_Motor_Z (void)`
This function reads the fault byte for the Z-axis h-bridge.
- `void DRV8830_Clear_Fault_Motor_X (void)`
This function clears the fault byte for the X-axis h-bridge.
- `void DRV8830_Clear_Fault_Motor_Y (void)`
This function clears the fault byte for the Y-axis h-bridge.
- `void DRV8830_Clear_Fault_Motor_Z (void)`
This function clears the fault byte for the Z-axis h-bridge.
- `uint8_t DRV8830_Check_Motors (void)`
This function checks to see if any of the 3 motors is falling.

3.52.1 Detailed Description

This file is code to read control the h-bridges.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 3/20/2015

Created: 11/8/2014 3:54:57 PM

Definition in file [DRV8830.h](#).

3.52.2 Macro Definition Documentation

3.52.2.1 #define Motor_Max_Voltage 4.98

The maximum voltage that we will allow for the h-bridge.

Definition at line 13 of file [DRV8830.h](#).

Referenced by [DRV8830_Voltage_To_Int\(\)](#), [Motor_Control_X_Open\(\)](#), [Motor_Control_Y_Open\(\)](#), [Motor_Control_Z\(\)](#), [State_Machine_State_3\(\)](#), [State_Machine_State_5\(\)](#), [State_Machine_Trasition_3\(\)](#), [State_Machine_Trasition_5\(\)](#), [Test_H_Bridge_Motor_ALL_Decrease\(\)](#), [Test_H_Bridge_Motor_ALL_Increase\(\)](#), [Test_H_Bridge_Motor_X_Decrease\(\)](#), [Test_H_Bridge_Motor_X_Increase\(\)](#), [Test_H_Bridge_Motor_Y_Decrease\(\)](#), [Test_H_Bridge_Motor_Y_Increase\(\)](#), [Test_H_Bridge_Motor_Z_Decrease\(\)](#), and [Test_H_Bridge_Motor_Z_Increase\(\)](#).

3.52.3 Function Documentation

3.52.3.1 uint8_t DRV8830_Check_Motors (void)

This function checks to see if any of the 3 motors is failing.

Returns

Whether or not the fault count is greater than a specified amount for each motor.

Bit 0: Z-axis motor

Bit 1: Y-axis motor

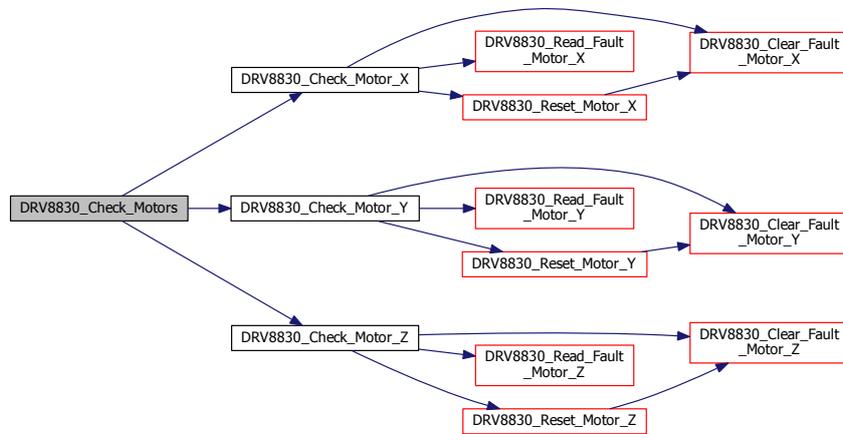
Bit 2: X-axis motor

Definition at line 703 of file [DRV8830.c](#).

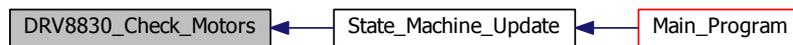
References [DRV8830_Check_Motor_X\(\)](#), [DRV8830_Check_Motor_Y\(\)](#), and [DRV8830_Check_Motor_Z\(\)](#).

Referenced by [State_Machine_Update\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.52.3.2 void DRV8830_Clear_Fault_Motor_X(void)

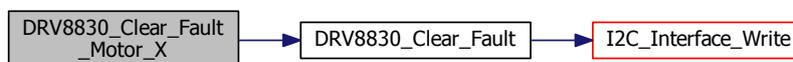
This function clears the fault byte for the X-axis h-bridge.

Definition at line 551 of file [DRV8830.c](#).

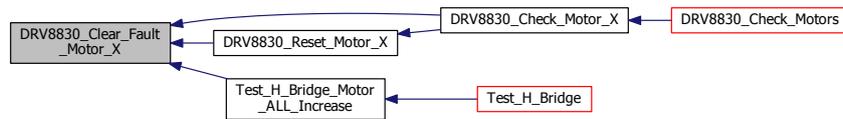
References [DRV8830_Clear_Fault\(\)](#), and [Motor_Address_X](#).

Referenced by [DRV8830_Check_Motor_X\(\)](#), [DRV8830_Reset_Motor_X\(\)](#), and [Test_H_Bridge_Motor_ALL_Increase\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.52.3.3 void DRV8830_Clear_Fault_Motor_Y (void)

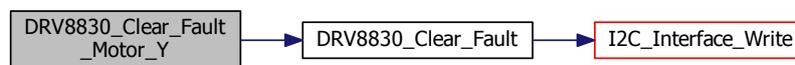
This function clears the fault byte for the Y-axis h-bridge.

Definition at line 560 of file [DRV8830.c](#).

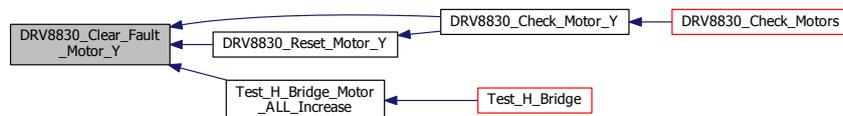
References [DRV8830_Clear_Fault\(\)](#), and [Motor_Address_Y](#).

Referenced by [DRV8830_Check_Motor_Y\(\)](#), [DRV8830_Reset_Motor_Y\(\)](#), and [Test_H_Bridge_Motor_ALL_Increase\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.52.3.4 void DRV8830_Clear_Fault_Motor_Z (void)

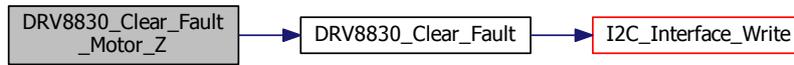
This function clears the fault byte for the Z-axis h-bridge.

Definition at line 569 of file [DRV8830.c](#).

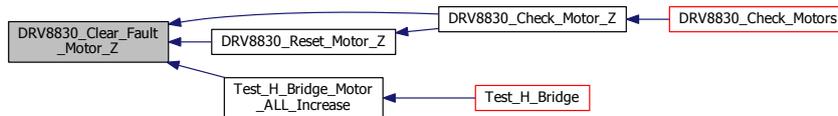
References [DRV8830_Clear_Fault\(\)](#), and [Motor_Address_Z](#).

Referenced by [DRV8830_Check_Motor_Z\(\)](#), [DRV8830_Reset_Motor_Z\(\)](#), and [Test_H_Bridge_Motor_ALL_Increase\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.52.3.5 void DRV8830_Initialize (void) [inline]

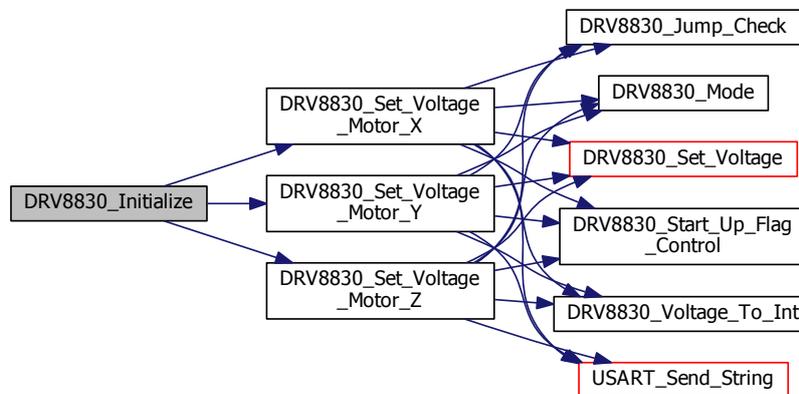
This function sets the initial output voltage for all three h-bridges.

Definition at line 487 of file DRV8830.c.

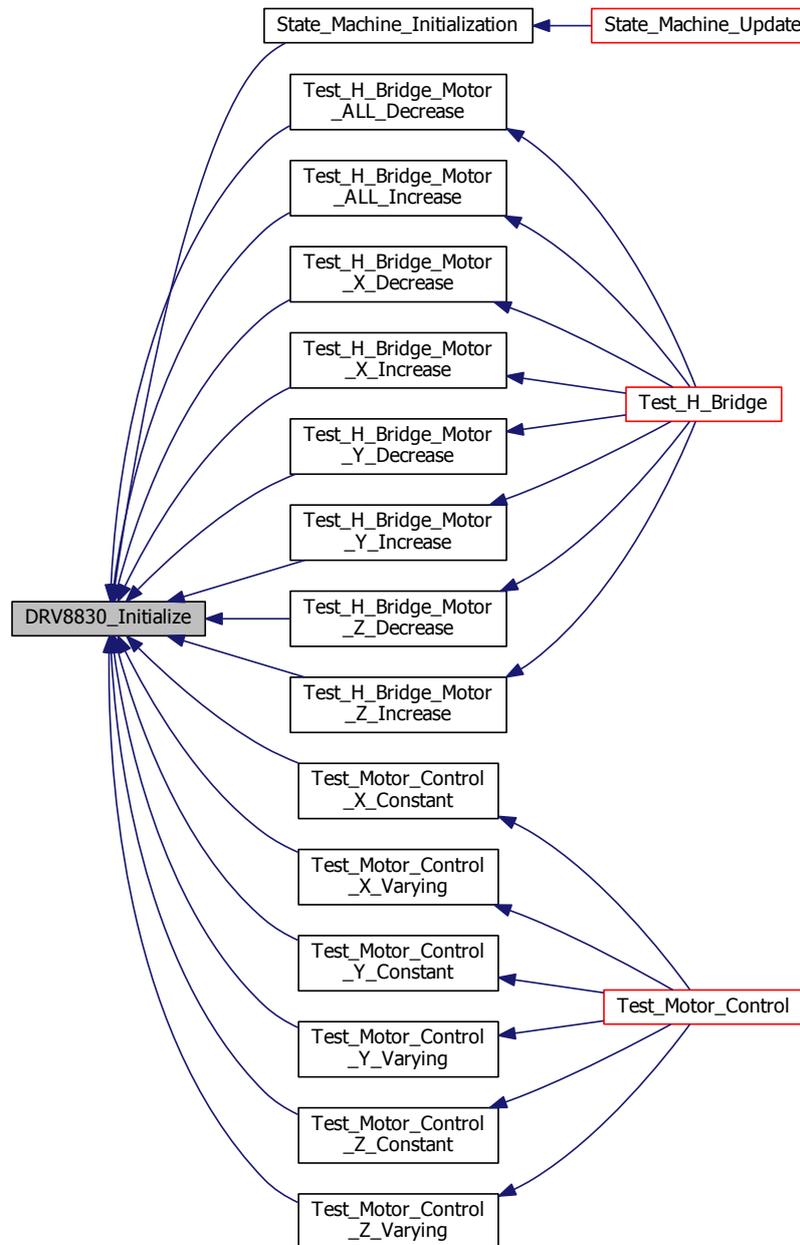
References DRV8830_Set_Voltage_Motor_X(), DRV8830_Set_Voltage_Motor_Y(), and DRV8830_Set_Voltage_Motor_Z().

Referenced by State_Machine_Initialization(), Test_H_Bridge_Motor_ALL_Decrease(), Test_H_Bridge_Motor_ALL_Increase(), Test_H_Bridge_Motor_X_Decrease(), Test_H_Bridge_Motor_X_Increase(), Test_H_Bridge_Motor_Y_Decrease(), Test_H_Bridge_Motor_Y_Increase(), Test_H_Bridge_Motor_Z_Decrease(), Test_H_Bridge_Motor_Z_Increase(), Test_Motor_Control_X_Constant(), Test_Motor_Control_X_Varying(), Test_Motor_Control_Y_Constant(), Test_Motor_Control_Y_Varying(), Test_Motor_Control_Z_Constant(), and Test_Motor_Control_Z_Varying().

Here is the call graph for this function:



Here is the caller graph for this function:



3.52.3.6 uint8_t DRV8830_Read_Fault_Motor_X (void)

This function reads the fault byte for the X-axis h-bridge.

Returns

The current fault byte for the X-axis h-bridge.

Definition at line 512 of file [DRV8830.c](#).

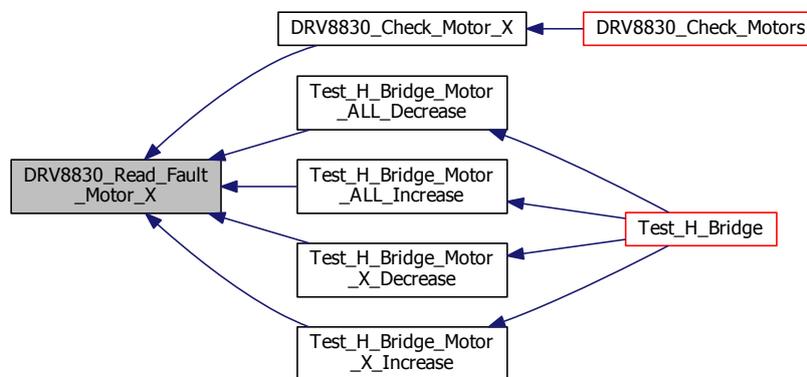
References [DRV8830_Read_Fault\(\)](#), and [Motor_Address_X](#).

Referenced by [DRV8830_Check_Motor_X\(\)](#), [Test_H_Bridge_Motor_ALL_Decrease\(\)](#), [Test_H_Bridge_Motor_ALL_Increase\(\)](#), [Test_H_Bridge_Motor_X_Decrease\(\)](#), and [Test_H_Bridge_Motor_X_Increase\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.52.3.7 uint8_t DRV8830_Read_Fault_Motor_Y (void)

This function reads the fault byte for the Y-axis h-bridge.

Returns

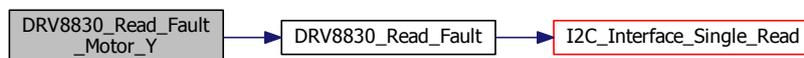
The current fault byte for the Y-axis h-bridge.

Definition at line 522 of file [DRV8830.c](#).

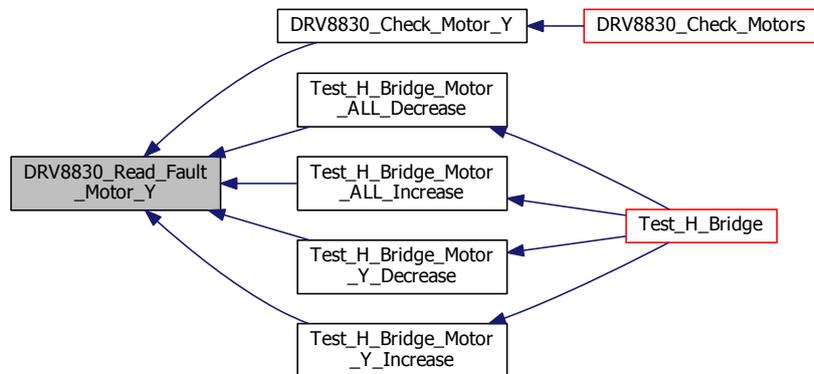
References [DRV8830_Read_Fault\(\)](#), and [Motor_Address_Y](#).

Referenced by [DRV8830_Check_Motor_Y\(\)](#), [Test_H_Bridge_Motor_ALL_Decrease\(\)](#), [Test_H_Bridge_Motor_ALL_Increase\(\)](#), [Test_H_Bridge_Motor_Y_Decrease\(\)](#), and [Test_H_Bridge_Motor_Y_Increase\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.52.3.8 uint8_t DRV8830_Read_Fault_Motor_Z (void)

This function reads the fault byte for the Z-axis h-bridge.

Returns

The current fault byte for the Z-axis h-bridge.

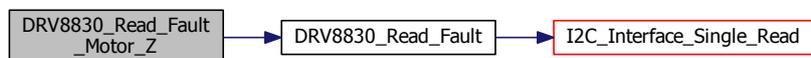
Return the fault byte for the Z-axis motor.

Definition at line 532 of file [DRV8830.c](#).

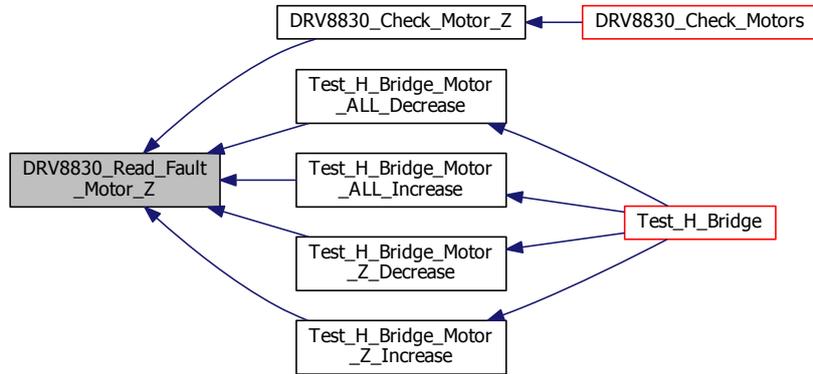
References [DRV8830_Read_Fault\(\)](#), and [Motor_Address_Z](#).

Referenced by [DRV8830_Check_Motor_Z\(\)](#), [Test_H_Bridge_Motor_ALL_Decrease\(\)](#), [Test_H_Bridge_Motor_ALL_Increase\(\)](#), [Test_H_Bridge_Motor_Z_Decrease\(\)](#), and [Test_H_Bridge_Motor_Z_Increase\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.52.3.9 void DRV8830_Set_Voltage_Motor_X (float Voltage)

This function sets the output voltage for the X-axis h-bridge. Note that this function only Updates the h-bridge if the mode or voltage is different.

Parameters

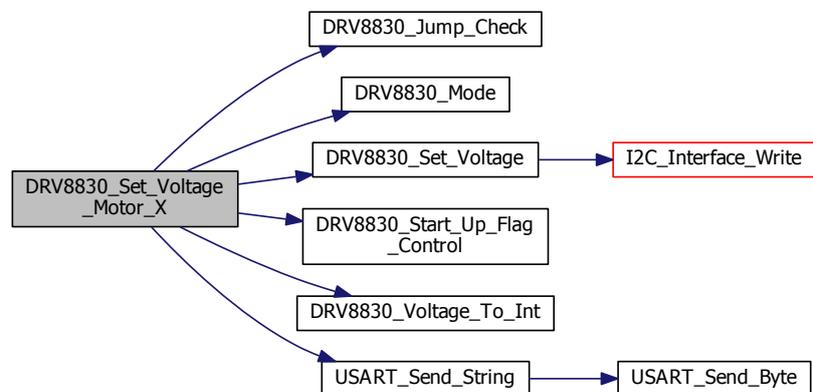
<i>Voltage</i>	The desired voltage for the motor to be at.
----------------	---

Definition at line 383 of file DRV8830.c.

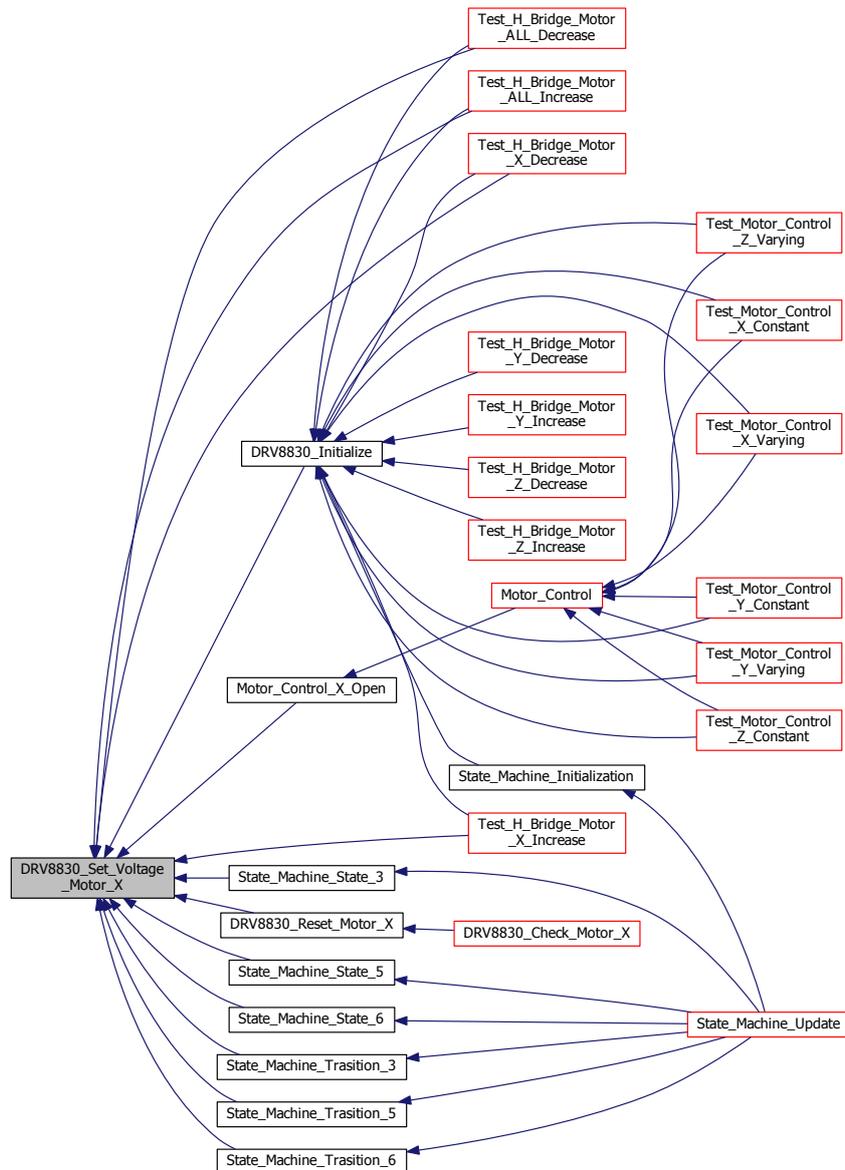
References DRV8830_Jump_Check(), DRV8830_Mode(), DRV8830_Set_Voltage(), DRV8830_Start_Up_Flag_Control(), DRV8830_Voltage_To_Int(), Motor_Address_X, and USART_Send_String().

Referenced by DRV8830_Initialize(), DRV8830_Reset_Motor_X(), Motor_Control_X_Open(), State_Machine_State_3(), State_Machine_State_5(), State_Machine_State_6(), State_Machine_Trasition_3(), State_Machine_Trasition_5(), State_Machine_Trasition_6(), Test_H_Bridge_Motor_ALL_Decrease(), Test_H_Bridge_Motor_ALL_Increase(), Test_H_Bridge_Motor_X_Decrease(), and Test_H_Bridge_Motor_X_Increase().

Here is the call graph for this function:



Here is the caller graph for this function:



3.52.3.10 void DRV8830_Set_Voltage_Motor_Y (float Voltage)

This function sets the output voltage for the Y-axis h-bridge. Note that this function only Updates the h-bridge if the mode or voltage is different.

Parameters

<i>Voltage</i>	The desired voltage for the motor to be at.
----------------	---

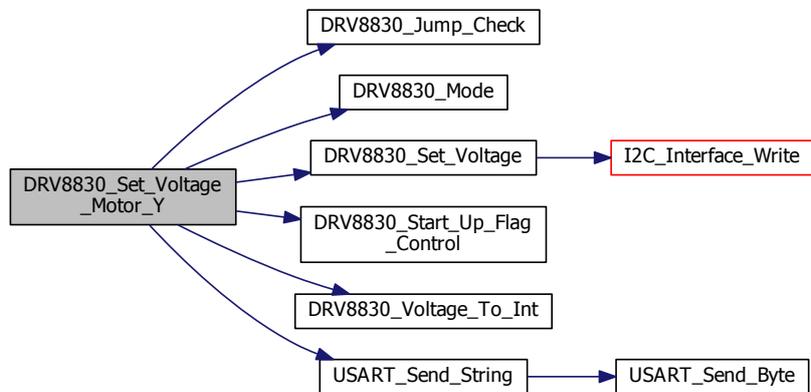
Definition at line 418 of file DRV8830.c.

References DRV8830_Jump_Check(), DRV8830_Mode(), DRV8830_Set_Voltage(), DRV8830_Start_Up_Flag_↔ Control(), DRV8830_Voltage_To_Int(), Motor_Address_Y, and USART_Send_String().

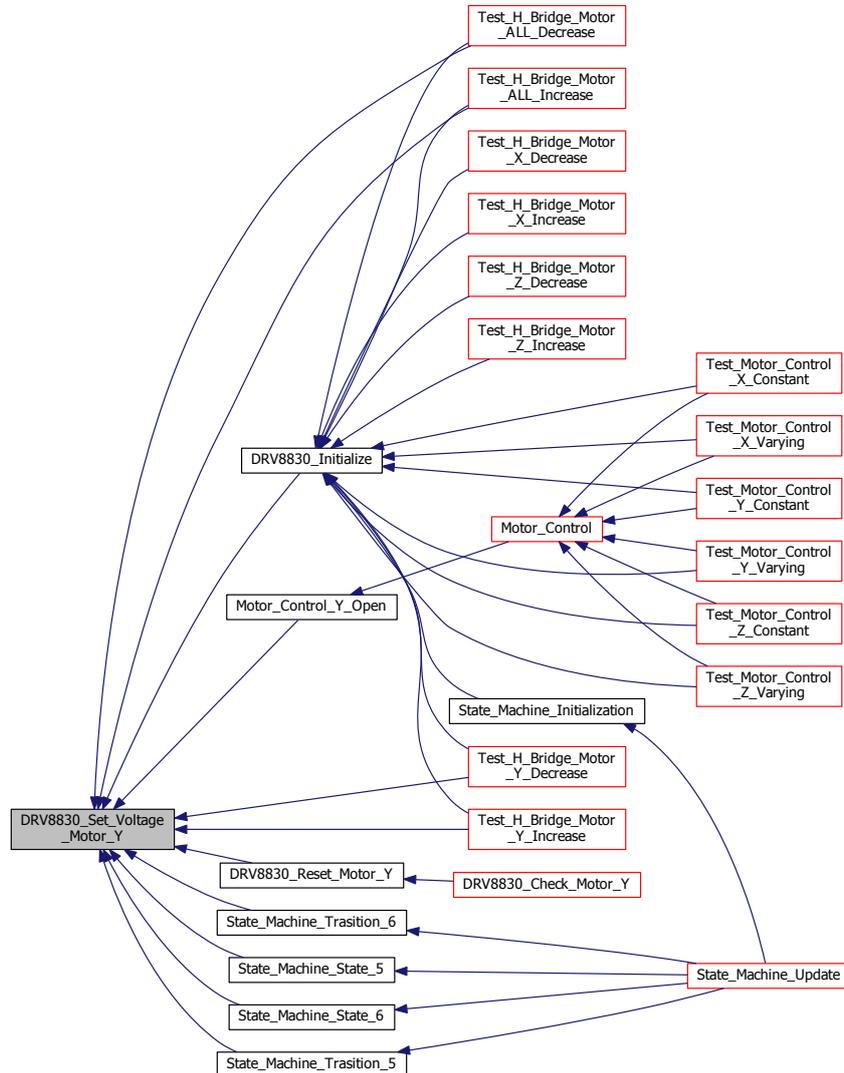
Referenced by DRV8830_Initialize(), DRV8830_Reset_Motor_Y(), Motor_Control_Y_Open(), State_Machine_↔

State_5(), State_Machine_State_6(), State_Machine_Trasition_5(), State_Machine_Trasition_6(), Test_H_Bridge↔_Motor_ALL_Decrease(), Test_H_Bridge_Motor_ALL_Increase(), Test_H_Bridge_Motor_Y_Decrease(), and Test↔_H_Bridge_Motor_Y_Increase().

Here is the call graph for this function:



Here is the caller graph for this function:



3.52.3.11 void DRV8830_Set_Voltage_Motor_Z (float Voltage)

This function sets the output voltage for the Z-axis h-bridge. Note that this function only Updates the h-bridge if the mode or voltage is different.

Parameters

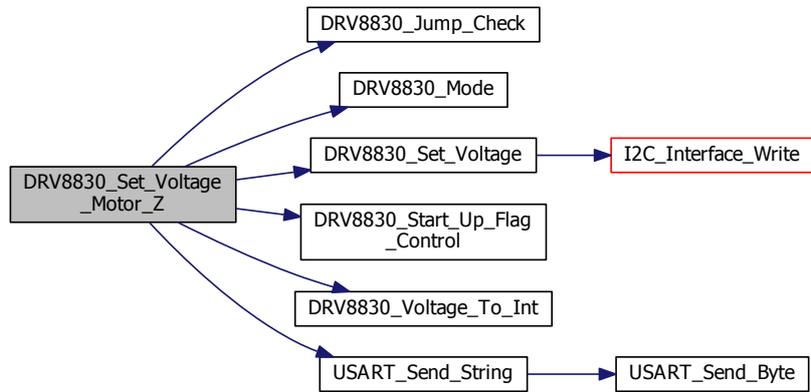
<i>Voltage</i>	The desired voltage for the motor to be at.
----------------	---

Definition at line 453 of file DRV8830.c.

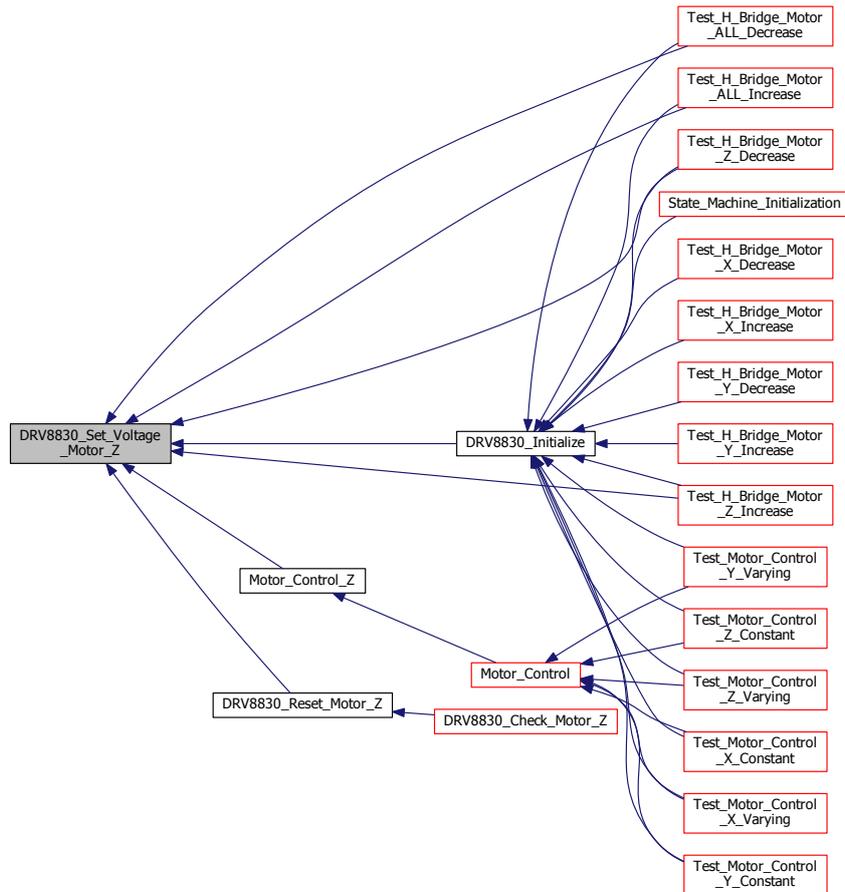
References [DRV8830_Jump_Check\(\)](#), [DRV8830_Mode\(\)](#), [DRV8830_Set_Voltage\(\)](#), [DRV8830_Start_Up_Flag_Control\(\)](#), [DRV8830_Voltage_To_Int\(\)](#), [Motor_Address_Z](#), and [USART_Send_String\(\)](#).

Referenced by [DRV8830_Initialize\(\)](#), [DRV8830_Reset_Motor_Z\(\)](#), [Motor_Control_Z\(\)](#), [Test_H_Bridge_Motor_ALL_Decrease\(\)](#), [Test_H_Bridge_Motor_ALL_Increase\(\)](#), [Test_H_Bridge_Motor_Z_Decrease\(\)](#), and [Test_H_Bridge_Motor_Z_Increase\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.53 DRV8830.h

```

00001 /**
00002  * @file DRV8830.h
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 3/20/2015\n Created: 11/8/2014 3:54:57 PM
00006  * @brief This file is code to read control the h-bridges.
00007  */
00008
00009 #ifndef DRV8830_H_
00010 #define DRV8830_H_
00011
00012 /** @brief The maximum voltage that we will allow for the h-bridge. */
00013 #define Motor_Max_Voltage 4.98
00014
00015 #include <stdint.h>
00016
00017 void DRV8830_Initialize(void);
00018 void DRV8830_Set_Voltage_Motor_X(float Voltage);
00019 void DRV8830_Set_Voltage_Motor_Y(float Voltage);
00020 void DRV8830_Set_Voltage_Motor_Z(float Voltage);
00021 uint8_t DRV8830_Read_Fault_Motor_X(void);
00022 uint8_t DRV8830_Read_Fault_Motor_Y(void);
00023 uint8_t DRV8830_Read_Fault_Motor_Z(void);
00024 void DRV8830_Clear_Fault_Motor_X(void);
00025 void DRV8830_Clear_Fault_Motor_Y(void);
00026 void DRV8830_Clear_Fault_Motor_Z(void);
00027 uint8_t DRV8830_Check_Motors(void);
00028
00029 #endif /* DRV8830_H_ */

```

3.54 Subsystem/IMU/LSM303.c File Reference

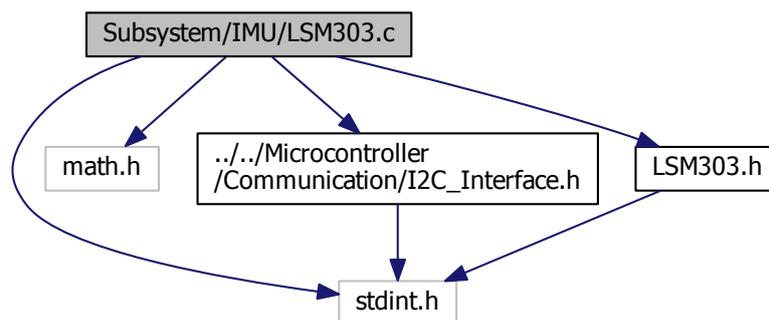
This file is code to control the LSM303.

```

#include <stdint.h>
#include <math.h>
#include "../Microcontroller/Communication/I2C_Interface.h"
#include "LSM303.h"

```

Include dependency graph for LSM303.c:



Macros

- #define Device_DLH 0
- #define Device_DLM 1
- #define Device_DLHC 2
- #define Device_D 3

- #define `Current_Device Device_D`
- #define `Register_Count 6`
- #define `Heading_Stable_Value 2`
- #define `TEST_REG_INVALID -1`
- #define `D_SA0_HIGH_ADDRESS 0x1D`
- #define `D_SA0_LOW_ADDRESS 0x1E`
- #define `NON_D_MAG_ADDRESS 0x1E`
- #define `NON_D_ACC_SA0_LOW_ADDRESS 0x18`
- #define `NON_D_ACC_SA0_HIGH_ADDRESS 0x19`
- #define `Radians_To_Degrees_Conv 57.2957795131`
- #define `X 0`
- #define `Y 1`
- #define `Z 2`
- #define `D_WHO_ID 0x49`
- #define `DLM_WHO_ID 0x3C`

Enumerations

- enum `Register_Address` {
`CTRL0 = 0x1F, CTRL1 = 0x20, CTRL_REG1_A = 0x20, CTRL2 = 0x21,`
`CTRL_REG2_A = 0x21, CTRL_REG4_A = 0x23, CTRL5 = 0x24, CTRL6 = 0x25,`
`CTRL7 = 0x26, OUT_X_L_A = 0x28, CRA_REG_M = 0x00, CRB_REG_M = 0x01,`
`MR_REG_M = 0x02, WHO_AM_I = 0x0F, OUT_X_H_M = 1, OUT_X_L_M = 2,`
`OUT_Y_H_M = 3, OUT_Y_L_M = 4, OUT_Z_H_M = 5, OUT_Z_L_M = 6,`
`DLH_OUT_X_H_M = 0x03, DLH_OUT_X_L_M = 0x04, DLH_OUT_Y_H_M = 0x05, DLH_OUT_Y_L_M =`
`0x06,`
`DLH_OUT_Z_H_M = 0x07, DLH_OUT_Z_L_M = 0x08, DLM_OUT_X_H_M = 0x03, DLM_OUT_X_L_M =`
`0x04,`
`DLM_OUT_Z_H_M = 0x05, DLM_OUT_Z_L_M = 0x06, DLM_OUT_Y_H_M = 0x07, DLM_OUT_Y_L_M =`
`0x08,`
`DLHC_OUT_X_H_M = 0x03, DLHC_OUT_X_L_M = 0x04, DLHC_OUT_Z_H_M = 0x05, DLHC_OUT_Z_L_M =`
`0x06,`
`DLHC_OUT_Y_H_M = 0x07, DLHC_OUT_Y_L_M = 0x08, D_OUT_X_L_M = 0x08, D_OUT_X_H_M = 0x09,`
`D_OUT_Y_L_M = 0x0A, D_OUT_Y_H_M = 0x0B, D_OUT_Z_L_M = 0x0C, D_OUT_Z_H_M = 0x0D }`

Functions

- `uint8_t LSM303_Test_Register (uint8_t Device_Address, uint8_t Register)`
- `void LSM303_Initialize ()`
- `void LSM303_Read_Mag (int16_t *Mag)`
- `void LSM303_Read_Acc (int16_t *Acc)`
- `void LSM303_Array_Cross_Product (float *a, float *b, float *Output)`
- `float LSM303_Array_Dot_Product (float *a, float *b)`
- `void LSM303_Array_Normalize (float *a)`
- `float LSM303_Heading (int16_t *Acc, int16_t *Mag)`
- `void LSM303_UpdateForwardHeading (void)`
- `float LSM303_HeadingWithOffset (float heading)`
- `uint8_t LSM303_RotationFinished (void)`
- `uint8_t LSM303_IsStablized (void)`

3.54.1 Detailed Description

This file is code to control the LSM303.

Author

Nicholas Sikkema

Version

Revision: 1.5

Date

Last Updated: 1/22/2015

Created: 10/30/2014 4:17:54 PM

Definition in file [LSM303.c](#).

3.54.2 Macro Definition Documentation

3.54.2.1 #define Current_Device Device_D

Definition at line 13 of file [LSM303.c](#).

3.54.2.2 #define D_SA0_HIGH_ADDRESS 0x1D

Definition at line 97 of file [LSM303.c](#).

Referenced by [LSM303_Initialize\(\)](#).

3.54.2.3 #define D_SA0_LOW_ADDRESS 0x1E

Definition at line 98 of file [LSM303.c](#).

Referenced by [LSM303_Initialize\(\)](#).

3.54.2.4 #define D_WHO_ID 0x49

Definition at line 108 of file [LSM303.c](#).

Referenced by [LSM303_Initialize\(\)](#).

3.54.2.5 #define Device_D 3

Definition at line 12 of file [LSM303.c](#).

3.54.2.6 #define Device_DLH 0

Definition at line 9 of file [LSM303.c](#).

3.54.2.7 #define Device_DLHC 2

Definition at line 11 of file [LSM303.c](#).

3.54.2.8 #define Device_DLM 1

Definition at line 10 of file [LSM303.c](#).

3.54.2.9 #define DLM_WHO_ID 0x3C

Definition at line 109 of file [LSM303.c](#).

3.54.2.10 #define Heading_Stable_Value 2

Definition at line 94 of file [LSM303.c](#).

Referenced by [LSM303_IsStablized\(\)](#).

3.54.2.11 #define NON_D_ACC_SA0_HIGH_ADDRESS 0x19

Definition at line 101 of file [LSM303.c](#).

Referenced by [LSM303_Initialize\(\)](#).

3.54.2.12 #define NON_D_ACC_SA0_LOW_ADDRESS 0x18

Definition at line 100 of file [LSM303.c](#).

Referenced by [LSM303_Initialize\(\)](#).

3.54.2.13 #define NON_D_MAG_ADDRESS 0x1E

Definition at line 99 of file [LSM303.c](#).

Referenced by [LSM303_Initialize\(\)](#).

3.54.2.14 #define Radians_To_Degrees_Conv 57.2957795131

Definition at line 102 of file [LSM303.c](#).

Referenced by [LSM303_Heading\(\)](#).

3.54.2.15 #define Register_Count 6

Definition at line 85 of file [LSM303.c](#).

3.54.2.16 #define TEST_REG_INVALID -1

Definition at line 96 of file [LSM303.c](#).

Referenced by [LSM303_Initialize\(\)](#), and [LSM303_Test_Register\(\)](#).

3.54.2.17 #define X 0

Definition at line 104 of file [LSM303.c](#).

Referenced by [LSM303_Array_Cross_Product\(\)](#), [LSM303_Array_Dot_Product\(\)](#), [LSM303_Array_Normalize\(\)](#), [LSM303_Heading\(\)](#), [LSM303_Read_Acc\(\)](#), [LSM303_Read_Mag\(\)](#), and [Test_Swarming\(\)](#).

3.54.2.18 #define Y 1

Definition at line 105 of file [LSM303.c](#).

Referenced by [LSM303_Array_Cross_Product\(\)](#), [LSM303_Array_Dot_Product\(\)](#), [LSM303_Array_Normalize\(\)](#), [LSM303_Heading\(\)](#), [LSM303_Read_Acc\(\)](#), [LSM303_Read_Mag\(\)](#), and [Test_Swarming\(\)](#).

3.54.2.19 #define Z 2

Definition at line 106 of file [LSM303.c](#).

Referenced by [LSM303_Array_Cross_Product\(\)](#), [LSM303_Array_Dot_Product\(\)](#), [LSM303_Array_Normalize\(\)](#), [LSM303_Heading\(\)](#), [LSM303_Read_Acc\(\)](#), and [LSM303_Read_Mag\(\)](#).

3.54.3 Enumeration Type Documentation

3.54.3.1 enum Register_Address

Enumerator

CTRL0
CTRL1
CTRL_REG1_A
CTRL2
CTRL_REG2_A
CTRL_REG4_A
CTRL5
CTRL6
CTRL7
OUT_X_L_A
CRA_REG_M
CRB_REG_M
MR_REG_M
WHO_AM_I
OUT_X_H_M
OUT_X_L_M
OUT_Y_H_M
OUT_Y_L_M
OUT_Z_H_M
OUT_Z_L_M
DLH_OUT_X_H_M
DLH_OUT_X_L_M
DLH_OUT_Y_H_M
DLH_OUT_Y_L_M
DLH_OUT_Z_H_M
DLH_OUT_Z_L_M
DLM_OUT_X_H_M
DLM_OUT_X_L_M
DLM_OUT_Z_H_M

DLM_OUT_Z_L_M
DLM_OUT_Y_H_M
DLM_OUT_Y_L_M
DLHC_OUT_X_H_M
DLHC_OUT_X_L_M
DLHC_OUT_Z_H_M
DLHC_OUT_Z_L_M
DLHC_OUT_Y_H_M
DLHC_OUT_Y_L_M
D_OUT_X_L_M
D_OUT_X_H_M
D_OUT_Y_L_M
D_OUT_Y_H_M
D_OUT_Z_L_M
D_OUT_Z_H_M

Definition at line 20 of file [LSM303.c](#).

3.54.4 Function Documentation

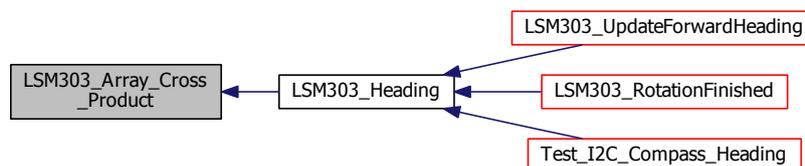
3.54.4.1 void `LSM303_Array_Cross_Product (float * a, float * b, float * Output)` [inline]

Definition at line 335 of file [LSM303.c](#).

References [X](#), [Y](#), and [Z](#).

Referenced by [LSM303_Heading\(\)](#).

Here is the caller graph for this function:



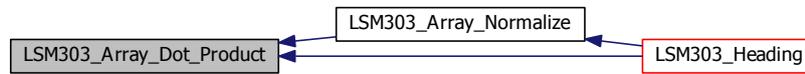
3.54.4.2 float `LSM303_Array_Dot_Product (float * a, float * b)` [inline]

Definition at line 342 of file [LSM303.c](#).

References [X](#), [Y](#), and [Z](#).

Referenced by [LSM303_Array_Normalize\(\)](#), and [LSM303_Heading\(\)](#).

Here is the caller graph for this function:



3.54.4.3 void LSM303_Array_Normalize (float * a) [inline]

Definition at line 347 of file [LSM303.c](#).

References [LSM303_Array_Dot_Product\(\)](#), [X](#), [Y](#), and [Z](#).

Referenced by [LSM303_Heading\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



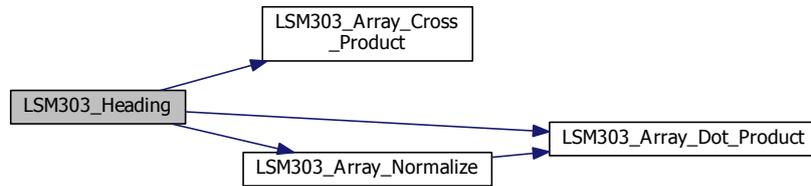
3.54.4.4 float LSM303_Heading (int16_t * Acc, int16_t * Mag)

Definition at line 355 of file [LSM303.c](#).

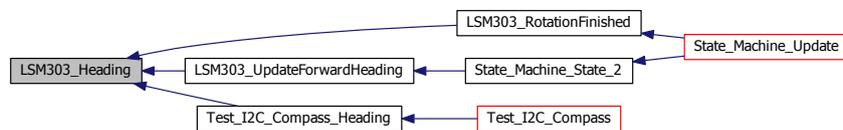
References [LSM303_Array_Cross_Product\(\)](#), [LSM303_Array_Dot_Product\(\)](#), [LSM303_Array_Normalize\(\)](#), [Radians_To_Degrees_Conv](#), [X](#), [Y](#), and [Z](#).

Referenced by [LSM303_RotationFinished\(\)](#), [LSM303_UpdateForwardHeading\(\)](#), and [Test_I2C_Compass_Heading\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

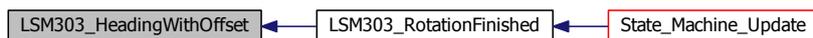


3.54.4.5 float LSM303_HeadingWithOffset (float heading)

Definition at line 383 of file [LSM303.c](#).

Referenced by [LSM303_RotationFinished\(\)](#).

Here is the caller graph for this function:



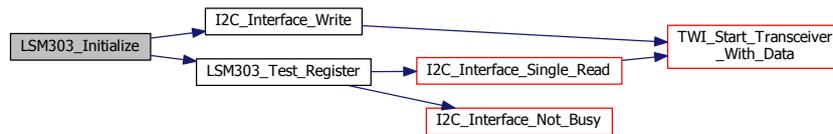
3.54.4.6 void LSM303_Initialize ()

Definition at line 124 of file [LSM303.c](#).

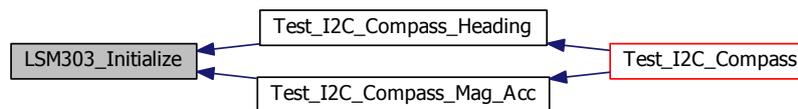
References [CRA_REG_M](#), [CRB_REG_M](#), [CTRL1](#), [CTRL2](#), [CTRL5](#), [CTRL6](#), [CTRL7](#), [CTRL_REG1_A](#), [CTRL_REG4_A](#), [D_OUT_X_H_M](#), [D_OUT_X_L_M](#), [D_OUT_Y_H_M](#), [D_OUT_Y_L_M](#), [D_OUT_Z_H_M](#), [D_OUT_Z_L_M](#), [D_SA0_HIGH_ADDRESS](#), [D_SA0_LOW_ADDRESS](#), [D_WHO_ID](#), [DLH_OUT_X_H_M](#), [DLH_OUT_X_L_M](#), [DLH_OUT_Y_H_M](#), [DLH_OUT_Y_L_M](#), [DLH_OUT_Z_H_M](#), [DLH_OUT_Z_L_M](#), [DLHC_OUT_X_H_M](#), [DLHC_OUT_X_L_M](#), [DLHC_OUT_Y_H_M](#), [DLHC_OUT_Y_L_M](#), [DLHC_OUT_Z_H_M](#), [DLHC_OUT_Z_L_M](#), [DLM_OUT_X_H_M](#), [DLM_OUT_X_L_M](#), [DLM_OUT_Y_H_M](#), [DLM_OUT_Y_L_M](#), [DLM_OUT_Z_H_M](#), [DLM_OUT_Z_L_M](#), [I2C_Interface_Write\(\)](#), [LSM303_Test_Register\(\)](#), [MR_REG_M](#), [NON_D_ACC_SA0_HIGH_ADDRESS](#), [NON_D_ACC_SA0_LOW_ADDRESS](#), [NON_D_MAG_ADDRESS](#), [OUT_X_H_M](#), [OUT_X_L_M](#), [OUT_Y_H_M](#), [OUT_Y_L_M](#), [OUT_Z_H_M](#), [OUT_Z_L_M](#), [TEST_REG_INVALID](#), and [WHO_AM_I](#).

Referenced by [Test_I2C_Compass_Heading\(\)](#), and [Test_I2C_Compass_Mag_Acc\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



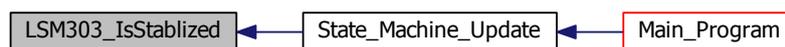
3.54.4.7 uint8_t LSM303_IsStablized (void)

Definition at line 416 of file [LSM303.c](#).

References [Heading_Stable_Value](#).

Referenced by [State_Machine_Update\(\)](#).

Here is the caller graph for this function:



3.54.4.8 void LSM303_Read_Acc (int16_t * Acc)

Definition at line 326 of file [LSM303.c](#).

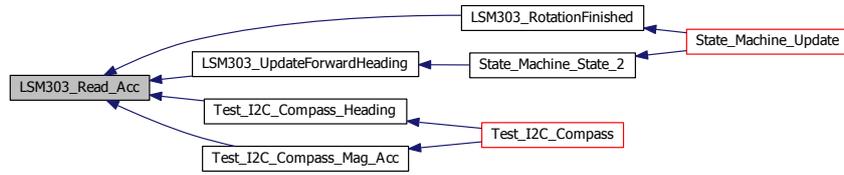
References [I2C_Interface_Read_Array\(\)](#), [OUT_X_L_A](#), [X](#), [Y](#), and [Z](#).

Referenced by [LSM303_RotationFinished\(\)](#), [LSM303_UpdateForwardHeading\(\)](#), [Test_I2C_Compass_Heading\(\)](#), and [Test_I2C_Compass_Mag_Acc\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



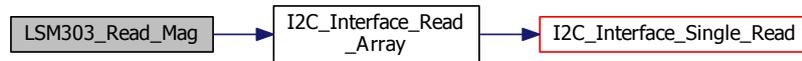
3.54.4.9 void LSM303_Read_Mag (int16_t * Mag)

Definition at line 301 of file LSM303.c.

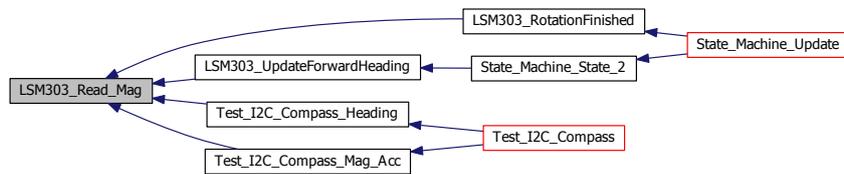
References I2C_Interface_Read_Array(), OUT_X_H_M, OUT_X_L_M, X, Y, and Z.

Referenced by LSM303_RotationFinished(), LSM303_UpdateForwardHeading(), Test_I2C_Compass_Heading(), and Test_I2C_Compass_Mag_Acc().

Here is the call graph for this function:



Here is the caller graph for this function:



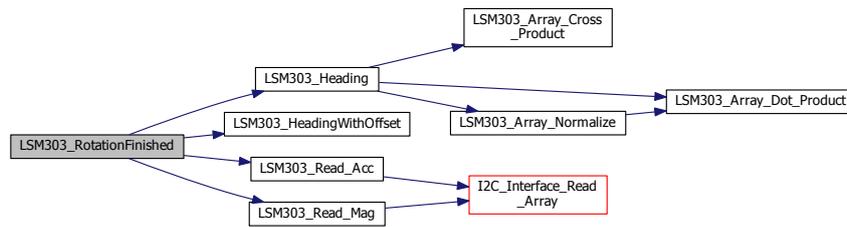
3.54.4.10 uint8_t LSM303_RotationFinished (void)

Definition at line 393 of file LSM303.c.

References LSM303_Heading(), LSM303_HeadingWithOffset(), LSM303_Read_Acc(), and LSM303_Read_Mag().

Referenced by State_Machine_Update().

Here is the call graph for this function:



Here is the caller graph for this function:



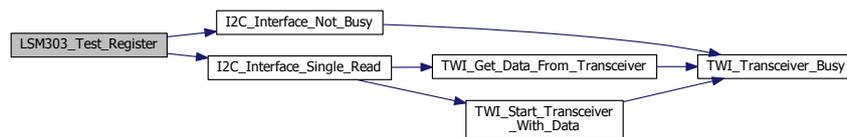
3.54.4.11 uint8_t LSM303_Test_Register (uint8_t Device_Address, uint8_t Register)

Definition at line 111 of file LSM303.c.

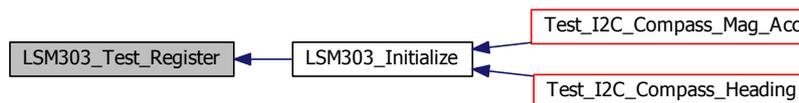
References [I2C_Interface_Not_Busy\(\)](#), [I2C_Interface_Single_Read\(\)](#), and [TEST_REG_INVALID](#).

Referenced by [LSM303_Initialize\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



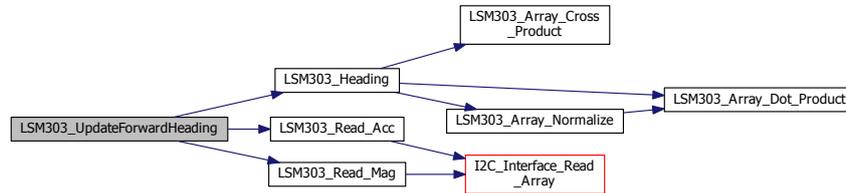
3.54.4.12 void LSM303_UpdateForwardHeading (void)

Definition at line 376 of file LSM303.c.

References [LSM303_Heading\(\)](#), [LSM303_Read_Acc\(\)](#), and [LSM303_Read_Mag\(\)](#).

Referenced by [State_Machine_State_2\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.55 LSM303.c

```

00001 /**
00002  * @file LSM303.c
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.5
00005  * @date Last Updated: 1/22/2015\n Created: 10/30/2014 4:17:54 PM
00006  * @brief This file is code to control the LSM303.
00007  */
00008
00009 #define Device_DLH 0
00010 #define Device_DLM 1
00011 #define Device_DLHC 2
00012 #define Device_D 3
00013 #define Current_Device Device_D
00014
00015 #include <stdint.h>
00016 #include <math.h>
00017 #include "../Microcontroller/Communication/I2C_Interface.h"
00018 #include "LSM303.h"
00019
00020 enum Register_Address
00021 {
00022     CTRL0           = 0x1F, // D
00023     CTRL1           = 0x20, // D
00024     CTRL_REG1_A    = 0x20, // DLH, DLM, DLHC
00025     CTRL2           = 0x21, // D
00026     CTRL_REG2_A    = 0x21, // DLH, DLM, DLHC
00027     CTRL_REG4_A    = 0x23, // DLH, DLM, DLHC
00028     CTRL5           = 0x24, // D
00029     CTRL6           = 0x25, // D
00030     CTRL7           = 0x26, // D
00031     OUT_X_L_A      = 0x28,
00032
00033     CRA_REG_M       = 0x00, // DLH, DLM, DLHC
00034     CRB_REG_M       = 0x01, // DLH, DLM, DLHC
00035     MR_REG_M        = 0x02, // DLH, DLM, DLHC
00036
00037     WHO_AM_I        = 0x0F, // D
00038
00039     // dummy addresses for registers in different locations on different devices;
00040     // the library translates these based on device type
00041     // value with sign flipped is used as index into translated_regs array
00042
00043     OUT_X_H_M       = 1,
00044     OUT_X_L_M       = 2,
00045     OUT_Y_H_M       = 3,
  
```

```

00046     OUT_Y_L_M           = 4,
00047     OUT_Z_H_M           = 5,
00048     OUT_Z_L_M           = 6,
00049     // Update dummy_reg_count if registers are added here!
00050
00051     // device-specific register addresses
00052
00053     DLH_OUT_X_H_M        = 0x03,
00054     DLH_OUT_X_L_M        = 0x04,
00055     DLH_OUT_Y_H_M        = 0x05,
00056     DLH_OUT_Y_L_M        = 0x06,
00057     DLH_OUT_Z_H_M        = 0x07,
00058     DLH_OUT_Z_L_M        = 0x08,
00059
00060     DLM_OUT_X_H_M        = 0x03,
00061     DLM_OUT_X_L_M        = 0x04,
00062     DLM_OUT_Z_H_M        = 0x05,
00063     DLM_OUT_Z_L_M        = 0x06,
00064     DLM_OUT_Y_H_M        = 0x07,
00065     DLM_OUT_Y_L_M        = 0x08,
00066
00067     DLHC_OUT_X_H_M       = 0x03,
00068     DLHC_OUT_X_L_M       = 0x04,
00069     DLHC_OUT_Z_H_M       = 0x05,
00070     DLHC_OUT_Z_L_M       = 0x06,
00071     DLHC_OUT_Y_H_M       = 0x07,
00072     DLHC_OUT_Y_L_M       = 0x08,
00073
00074     D_OUT_X_L_M          = 0x08,
00075     D_OUT_X_H_M          = 0x09,
00076     D_OUT_Y_L_M          = 0x0A,
00077     D_OUT_Y_H_M          = 0x0B,
00078     D_OUT_Z_L_M          = 0x0C,
00079     D_OUT_Z_H_M          = 0x0D
00080 };
00081
00082 //Global Variables
00083 static uint8_t Acc_Address = 0;
00084 static uint8_t Mag_Address = 0;
00085 #define Register_Count 6
00086 static enum Register_Address Translated_Register[Register_Count+1]; // index 0
00087     not used
00087 static float Mag_Min[3] = {-32767, -32767, -32767};
00088 static float Mag_Max[3] = {32767, 32767, 32767};
00089 static float Mag_From[3] = {0,0,0};
00090 static int16_t Mag_Temp[3] = {0,0,0};
00091 static int16_t Acc_Temp[3] = {0,0,0};
00092 static float Forward_Heading = 0;
00093
00094 #define Heading_Stable_Value 2
00095
00096 #define TEST_REG_INVALID          -1
00097 #define D_SAO_HIGH_ADDRESS        0x1D // D with SA0 high
00098 #define D_SAO_LOW_ADDRESS         0x1E // D with SA0 low or non-D magnetometer
00099 #define NON_D_MAG_ADDRESS         0x1E // D with SA0 low or non-D magnetometer
00100 #define NON_D_ACC_SAO_LOW_ADDRESS 0x18 // non-D accelerometer with SA0 low
00101 #define NON_D_ACC_SAO_HIGH_ADDRESS 0x19 // non-D accelerometer with SA0 high
00102 #define Radians_To_Degrees_Conv  57.2957795131
00103
00104 #define X 0
00105 #define Y 1
00106 #define Z 2
00107
00108 #define D_WHO_ID    0x49
00109 #define DLM_WHO_ID  0x3C
00110
00111 uint8_t LSM303_Test_Register(uint8_t Device_Address, uint8_t Register)
00112 {
00113     uint8_t Message_Buff = I2C_Interface_Single_Read(Device_Address, Register);
00114     if(I2C_Interface_Not_Busy())
00115     {
00116         return Message_Buff;
00117     }
00118     else
00119     {
00120         return TEST_REG_INVALID;
00121     }
00122 }
00123
00124 void LSM303_Initialize()
00125 {
00126     // determine Device type if necessary
00127     #if Current_Device == Device_D
00128         if(LSM303_Test_Register(D_SAO_HIGH_ADDRESS,
00129             WHO_AM_I) == D_WHO_ID)
00129         {
00130             Acc_Address = D_SAO_HIGH_ADDRESS;

```

```

00131     Mag_Address = D_SA0_HIGH_ADDRESS;
00132 }
00133 else
00134 {
00135     Acc_Address = D_SA0_LOW_ADDRESS;
00136     Mag_Address = D_SA0_LOW_ADDRESS;
00137 }
00138
00139 // set device addresses and translated register addresses
00140 Translated_Register[OUT_X_L_M] = D_OUT_X_L_M;
00141 Translated_Register[OUT_X_H_M] = D_OUT_X_H_M;
00142 Translated_Register[OUT_Y_L_M] = D_OUT_Y_L_M;
00143 Translated_Register[OUT_Y_H_M] = D_OUT_Y_H_M;
00144 Translated_Register[OUT_Z_L_M] = D_OUT_Z_L_M;
00145 Translated_Register[OUT_Z_H_M] = D_OUT_Z_H_M;
00146
00147 Mag_From[0] = 1;
00148 Mag_From[1] = 0;
00149 Mag_From[2] = 0;
00150
00151 // Accelerometer
00152
00153 // 0x57 = 0b01010111
00154 // AFS = 0 (+/- 2 g full scale)
00155 I2C_Interface_Write(Acc_Address, CTRL2, 0x00);
00156
00157 // 0x57 = 0b01010111
00158 // AODR = 0101 (50 Hz ODR);AZEN = AYEN = AXEN = 1 (all axes enabled)
00159 I2C_Interface_Write(Acc_Address, CTRL1, 0x57);
00160
00161 // Magnetometer
00162
00163 // 0x64 = 0b01100100
00164 // M_RES = 11 (high resolution mode);M_ODR = 001 (6.25 Hz ODR)
00165 I2C_Interface_Write(Mag_Address, CTRL5, 0x64);
00166
00167 // 0x20 = 0b00100000
00168 // MFS = 01 (+/- 4 gauss full scale)
00169 I2C_Interface_Write(Mag_Address, CTRL6, 0x20);
00170
00171 // 0x00 = 0b00000000
00172 // MLP = 0 (low power mode off);MD = 00 (continuous-conversion mode)
00173 I2C_Interface_Write(Mag_Address, CTRL7, 0x00);
00174 #elif Current_Device == Device_DLHC
00175 // set device addresses and translated register addresses
00176 Acc_Address = NON_D_ACC_SA0_HIGH_ADDRESS;// DLHC doesn't have SA0 but
uses same acc address as DLH/DLM with SA0 high
00177 Mag_Address = NON_D_MAG_ADDRESS;
00178 Translated_Register[OUT_X_H_M] = DLHC_OUT_X_H_M;
00179 Translated_Register[OUT_X_L_M] = DLHC_OUT_X_L_M;
00180 Translated_Register[OUT_Y_H_M] = DLHC_OUT_Y_H_M;
00181 Translated_Register[OUT_Y_L_M] = DLHC_OUT_Y_L_M;
00182 Translated_Register[OUT_Z_H_M] = DLHC_OUT_Z_H_M;
00183 Translated_Register[OUT_Z_L_M] = DLHC_OUT_Z_L_M;
00184
00185 Mag_From[0] = 0;
00186 Mag_From[1] = -1;
00187 Mag_From[2] = 0;
00188
00189 // Accelerometer
00190
00191 // 0x08 = 0b00001000
00192 // FS = 00 (+/- 2 g full scale);HR = 1 (high resolution enable)
00193 I2C_Interface_Write(Acc_Address, CTRL_REG4_A, 0x08);
00194
00195 // 0x47 = 0b01000111
00196 // ODR = 0100 (50 Hz ODR);LPen = 0 (normal mode);Zen = Yen = Xen = 1 (all axes enabled)
00197 I2C_Interface_Write(Acc_Address, CTRL_REG1_A, 0x47);
00198
00199 // Magnetometer
00200
00201 // 0x0C = 0b00001100
00202 // DO = 011 (7.5 Hz ODR)
00203 I2C_Interface_Write(Mag_Address, CRA_REG_M, 0x0C);
00204
00205 // 0x20 = 0b00100000
00206 // GN = 001 (+/- 1.3 gauss full scale)
00207 I2C_Interface_Write(Mag_Address, CRB_REG_M, 0x20);
00208
00209 // 0x00 = 0b00000000
00210 // MD = 00 (continuous-conversion mode)
00211 I2C_Interface_Write(Mag_Address, MR_REG_M, 0x00);
00212 #elif Current_Device == Device_DLM
00213 // Device responds to address 0011000 with DLM ID;guess that it's a DLM
00214 if(LSM303_Test_Register(NON_D_ACC_SA0_LOW_ADDRESS,
CTRL_REG1_A) != TEST_REG_INVALID)
00215 {

```

```

00216     Acc_Address = NON_D_ACC_SAO_HIGH_ADDRESS;
00217 }
00218 else
00219 {
00220     Acc_Address = NON_D_ACC_SAO_LOW_ADDRESS;
00221 }
00222
00223 // set device addresses and translated register addresses
00224 Mag_Address = NON_D_MAG_ADDRESS;
00225 Translated_Register[OUT_X_H_M] = DLH_OUT_X_H_M;
00226 Translated_Register[OUT_X_L_M] = DLH_OUT_X_L_M;
00227 Translated_Register[OUT_Y_H_M] = DLH_OUT_Y_H_M;
00228 Translated_Register[OUT_Y_L_M] = DLH_OUT_Y_L_M;
00229 Translated_Register[OUT_Z_H_M] = DLH_OUT_Z_H_M;
00230 Translated_Register[OUT_Z_L_M] = DLH_OUT_Z_L_M;
00231
00232 Mag_From[0] = 0;
00233 Mag_From[1] = -1;
00234 Mag_From[2] = 0;
00235
00236 // Accelerometer
00237
00238 // 0x00 = 0b00000000
00239 // FS = 00 (+/- 2 g full scale)
00240 I2C_Interface_Write(Acc_Address, CTRL_REG4_A, 0x00);
00241
00242 // 0x27 = 0b00100111
00243 // PM = 001 (normal mode);DR = 00 (50 Hz ODR);Zen = Yen = Xen = 1 (all axes enabled)
00244 I2C_Interface_Write(Acc_Address, CTRL_REG1_A, 0x27);
00245
00246 // Magnetometer
00247
00248 // 0x0C = 0b00001100
00249 // DO = 011 (7.5 Hz ODR)
00250 I2C_Interface_Write(Mag_Address, CRA_REG_M, 0x0C);
00251
00252 // 0x20 = 0b00100000
00253 // GN = 001 (+/- 1.3 gauss full scale)
00254 I2C_Interface_Write(Mag_Address, CRB_REG_M, 0x20);
00255
00256 // 0x00 = 0b00000000
00257 // MD = 00 (continuous-conversion mode)
00258 I2C_Interface_Write(Mag_Address, MR_REG_M, 0x00);
00259 #elif Current_Device == Device_DLM
00260 // set device addresses and translated register addresses
00261 Acc_Address = NON_D_ACC_SAO_HIGH_ADDRESS;
00262 Mag_Address = NON_D_MAG_ADDRESS;
00263 Translated_Register[OUT_X_H_M] = DLM_OUT_X_H_M;
00264 Translated_Register[OUT_X_L_M] = DLM_OUT_X_L_M;
00265 Translated_Register[OUT_Y_H_M] = DLM_OUT_Y_H_M;
00266 Translated_Register[OUT_Y_L_M] = DLM_OUT_Y_L_M;
00267 Translated_Register[OUT_Z_H_M] = DLM_OUT_Z_H_M;
00268 Translated_Register[OUT_Z_L_M] = DLM_OUT_Z_L_M;
00269
00270 Mag_From[0] = 0;
00271 Mag_From[1] = -1;
00272 Mag_From[2] = 0;
00273
00274 // Accelerometer
00275
00276 // 0x00 = 0b00000000
00277 // FS = 00 (+/- 2 g full scale)
00278 I2C_Interface_Write(Acc_Address, CTRL_REG4_A, 0x00);
00279
00280 // 0x27 = 0b00100111
00281 // PM = 001 (normal mode);DR = 00 (50 Hz ODR);Zen = Yen = Xen = 1 (all axes enabled)
00282 I2C_Interface_Write(Acc_Address, CTRL_REG1_A, 0x27);
00283
00284 // Magnetometer
00285
00286 // 0x0C = 0b00001100
00287 // DO = 011 (7.5 Hz ODR)
00288 I2C_Interface_Write(Mag_Address, CRA_REG_M, 0x0C);
00289
00290 // 0x20 = 0b00100000
00291 // GN = 001 (+/- 1.3 gauss full scale)
00292 I2C_Interface_Write(Mag_Address, CRB_REG_M, 0x20);
00293
00294 // 0x00 = 0b00000000
00295 // MD = 00 (continuous-conversion mode)
00296 I2C_Interface_Write(Mag_Address, MR_REG_M, 0x00);
00297 #endif
00298 }
00299
00300
00301 void LSM303_Read_Mag(int16_t *Mag)
00302 {

```

```

00303     uint8_t Readings[6] = {0,0,0,0,0,0};
00304     #if Current_Device == Device_D
00305         I2C_Interface_Read_Array(Mag_Address,Translated_Register[
OUT_X_L_M]|(1<<7),Readings,6);
00306     #else
00307         I2C_Interface_Read_Array(Mag_Address,Translated_Register[
OUT_X_H_M],Readings,6);
00308     #endif
00309
00310     #if (Current_Device == Device_D)
00311         Mag[X] = (Readings[1]<<8|Readings[0]);
00312         Mag[Y] = (Readings[3]<<8|Readings[2]);
00313         Mag[Z] = (Readings[5]<<8|Readings[4]);
00314     #elif(Current_Device == Device_DLH)
00315         Mag[X] = (Readings[0]<<8|Readings[1]);
00316         Mag[Y] = (Readings[2]<<8|Readings[3]);
00317         Mag[Z] = (Readings[4]<<8|Readings[5]);
00318     #else
00319         Mag[X] = (Readings[0]<<8|Readings[1]);
00320         Mag[Y] = (Readings[4]<<8|Readings[5]);
00321         Mag[Z] = (Readings[2]<<8|Readings[3]);
00322     #endif
00323 }
00324
00325
00326 void LSM303_Read_Acc(int16_t *Acc)
00327 {
00328     uint8_t Readings[6] = {0,0,0,0,0,0};
00329     I2C_Interface_Read_Array(Acc_Address,OUT_X_L_A,Readings,6);
00330     Acc[X] = (Readings[1]<<8|Readings[0]);
00331     Acc[Y] = (Readings[3]<<8|Readings[2]);
00332     Acc[Z] = (Readings[5]<<8|Readings[4]);
00333 }
00334
00335 inline void LSM303_Array_Cross_Product(float *a, float *b, float *Output)
00336 {
00337     Output[X] = (a[Y]*b[Z]) - (a[Z]*b[Y]);
00338     Output[Y] = (a[Z]*b[X]) - (a[X]*b[Z]);
00339     Output[Z] = (a[X]*b[Y]) - (a[Y]*b[X]);
00340 }
00341
00342 inline float LSM303_Array_Dot_Product(float *a, float *b)
00343 {
00344     return (a[X]*b[X]+a[Y]*b[Y]+a[Z]*b[Z]);
00345 }
00346
00347 inline void LSM303_Array_Normalize(float *a)
00348 {
00349     float Magnitude = sqrt(LSM303_Array_Dot_Product(a,a));
00350     a[X] /= Magnitude;
00351     a[Y] /= Magnitude;
00352     a[Z] /= Magnitude;
00353 }
00354
00355 float LSM303_Heading(int16_t *Acc, int16_t *Mag)
00356 {
00357     Mag[X] -= (Mag_Min[X]+Mag_Max[X])/2;
00358     Mag[Y] -= (Mag_Min[Y]+Mag_Max[Y])/2;
00359     Mag[Z] -= (Mag_Min[Z]+Mag_Max[Z])/2;
00360     float E[3] = {0,0,0};
00361     float N[3] = {0,0,0};
00362     float TempMag[3] = {Mag[X], Mag[Y], Mag[Z]};
00363     float TempAcc[3] = {Acc[X], Acc[Y], Acc[Z]};
00364     LSM303_Array_Cross_Product(TempMag,TempAcc,E);
00365     LSM303_Array_Normalize(E);
00366     LSM303_Array_Cross_Product(TempAcc,E,N);
00367     LSM303_Array_Normalize(N);
00368     float heading = atan2(LSM303_Array_Dot_Product(E,Mag_From),
LSM303_Array_Dot_Product(N,Mag_From))*
Radians_To_Degrees_Conv;
00369     if(heading < 0)
00370     {
00371         heading += 360;
00372     }
00373     return heading;
00374 }
00375
00376 void LSM303_UpdateForwardHeading(void)
00377 {
00378     LSM303_Read_Mag(Mag_Temp);
00379     LSM303_Read_Acc(Acc_Temp);
00380     Forward_Heading = LSM303_Heading(Acc_Temp, Mag_Temp);
00381 }
00382
00383 float LSM303_HeadingWithOffset(float heading)
00384 {
00385     float Temp_Heading = heading - Forward_Heading;

```

```

00386     if (Temp_Heading < 0)
00387     {
00388         Temp_Heading += 360;
00389     }
00390     return Temp_Heading;
00391 }
00392
00393 uint8_t LSM303_RotationFinished(void)
00394 {
00395     LSM303_Read_Mag (Mag_Temp);
00396     LSM303_Read_Acc (Acc_Temp);
00397     float Heading = LSM303_HeadingWithOffset (
LSM303_Heading (Acc_Temp, Mag_Temp));
00398     static float Previous_Heading = 0;
00399     static uint8_t flag = 0;
00400     int Return_Value = 0;
00401     if ((Heading<5&&Heading>-5)&&(Previous_Heading>5||Previous_Heading<-5))
00402     {
00403         if (flag == 0)
00404         {
00405             flag = 1;
00406         }
00407         else
00408         {
00409             Return_Value = 1;
00410         }
00411     }
00412     Previous_Heading = Heading;
00413     return (Return_Value);
00414 }
00415
00416 uint8_t LSM303_IsStablized(void)
00417 {
00418     static float Previous_Heading_Value = 1000;
00419     float Heading_Change = Forward_Heading-Previous_Heading_Value;
00420     Previous_Heading_Value = Forward_Heading;
00421     if (Heading_Change<Heading_Stable_Value||Heading_Change<-
Heading_Stable_Value)
00422     {
00423         return(1);
00424     }
00425     else
00426     {
00427         return(0);
00428     }
00429 }

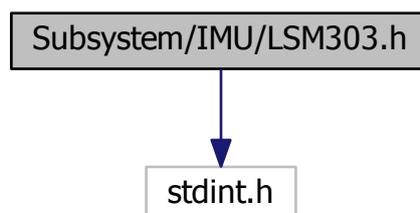
```

3.56 Subsystem/IMU/LSM303.h File Reference

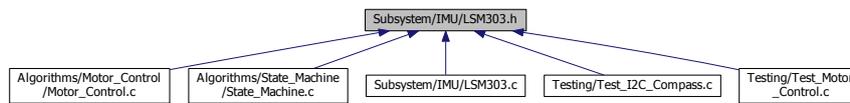
This file is code to control the LSM303.

```
#include <stdint.h>
```

Include dependency graph for LSM303.h:



This graph shows which files directly or indirectly include this file:



Functions

- void [LSM303_Initialize](#) ()
- void [LSM303_Read_Mag](#) (int16_t *Mag)
- void [LSM303_Read_Acc](#) (int16_t *Acc)
- float [LSM303_Heading](#) (int16_t *Acc, int16_t *Mag)
- void [LSM303_UpdateForwardHeading](#) (void)
- float [LSM303_HeadingWithOffset](#) (float heading)
- uint8_t [LSM303_RotationFinished](#) (void)
- uint8_t [LSM303_IsStablized](#) (void)

3.56.1 Detailed Description

This file is code to control the LSM303.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 1/22/2015
Created: 10/30/2014 4:42:22 PM

Definition in file [LSM303.h](#).

3.56.2 Function Documentation

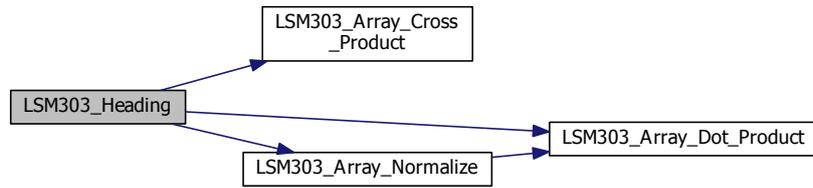
3.56.2.1 float [LSM303_Heading](#) (int16_t * *Acc*, int16_t * *Mag*)

Definition at line 355 of file [LSM303.c](#).

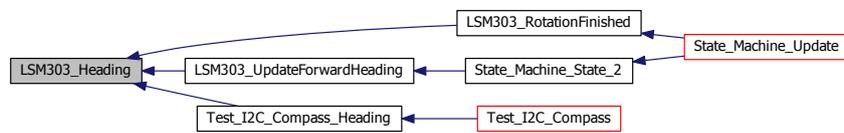
References [LSM303_Array_Cross_Product\(\)](#), [LSM303_Array_Dot_Product\(\)](#), [LSM303_Array_Normalize\(\)](#), [Radians_To_Degrees_Conv](#), [X](#), [Y](#), and [Z](#).

Referenced by [LSM303_RotationFinished\(\)](#), [LSM303_UpdateForwardHeading\(\)](#), and [Test_I2C_Compass_Heading\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

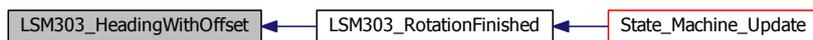


3.56.2.2 float LSM303_HeadingWithOffset (float heading)

Definition at line 383 of file LSM303.c.

Referenced by LSM303_RotationFinished().

Here is the caller graph for this function:



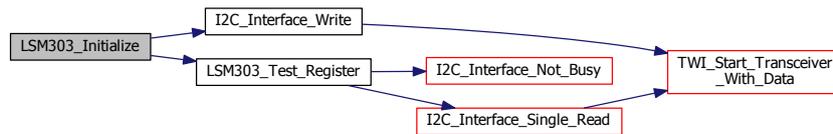
3.56.2.3 void LSM303_Initialize ()

Definition at line 124 of file LSM303.c.

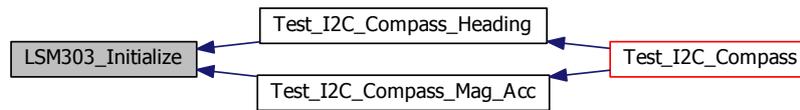
References CRA_REG_M, CRB_REG_M, CTRL1, CTRL2, CTRL5, CTRL6, CTRL7, CTRL_REG1_A, CTRL_REG4_A, D_OUT_X_H_M, D_OUT_X_L_M, D_OUT_Y_H_M, D_OUT_Y_L_M, D_OUT_Z_H_M, D_OUT_Z_L_M, D_SA0_HIGH_ADDRESS, D_SA0_LOW_ADDRESS, D_WHO_ID, DLH_OUT_X_H_M, DLH_OUT_X_L_M, DLH_OUT_Y_H_M, DLH_OUT_Y_L_M, DLH_OUT_Z_H_M, DLH_OUT_Z_L_M, DLHC_OUT_X_H_M, DLHC_OUT_X_L_M, DLHC_OUT_Y_H_M, DLHC_OUT_Y_L_M, DLHC_OUT_Z_H_M, DLHC_OUT_Z_L_M, DLM_OUT_X_H_M, DLM_OUT_X_L_M, DLM_OUT_Y_H_M, DLM_OUT_Y_L_M, DLM_OUT_Z_H_M, DLM_OUT_Z_L_M, I2C_Interface_Write(), LSM303_Test_Register(), MR_REG_M, NON_D_ACC_SA0_HIGH_ADDRESS, NON_D_ACC_SA0_LOW_ADDRESS, NON_D_MAG_ADDRESS, OUT_X_H_M, OUT_X_L_M, OUT_Y_H_M, OUT_Y_L_M, OUT_Z_H_M, OUT_Z_L_M, TEST_REG_INVALID, and WHO_AM_I.

Referenced by Test_I2C_Compass_Heading(), and Test_I2C_Compass_Mag_Acc().

Here is the call graph for this function:



Here is the caller graph for this function:



3.56.2.4 uint8_t LSM303_IsStablized (void)

Definition at line 416 of file [LSM303.c](#).

References [Heading_Stable_Value](#).

Referenced by [State_Machine_Update\(\)](#).

Here is the caller graph for this function:



3.56.2.5 void LSM303_Read_Acc (int16_t * Acc)

Definition at line 326 of file [LSM303.c](#).

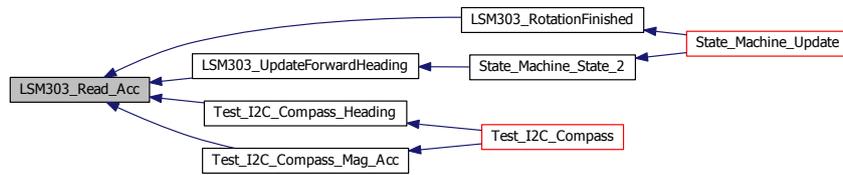
References [I2C_Interface_Read_Array\(\)](#), [OUT_X_L_A](#), [X](#), [Y](#), and [Z](#).

Referenced by [LSM303_RotationFinished\(\)](#), [LSM303_UpdateForwardHeading\(\)](#), [Test_I2C_Compass_Heading\(\)](#), and [Test_I2C_Compass_Mag_Acc\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.56.2.6 void LSM303_Read_Mag (int16_t * Mag)

Definition at line 301 of file [LSM303.c](#).

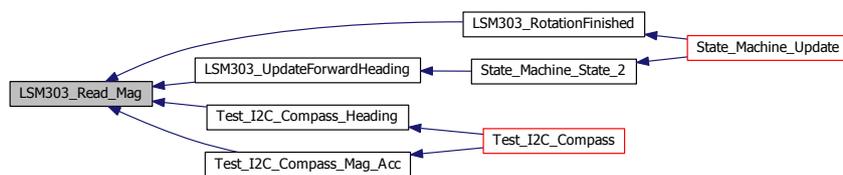
References [I2C_Interface_Read_Array\(\)](#), [OUT_X_H_M](#), [OUT_X_L_M](#), [X](#), [Y](#), and [Z](#).

Referenced by [LSM303_RotationFinished\(\)](#), [LSM303_UpdateForwardHeading\(\)](#), [Test_I2C_Compass_Heading\(\)](#), and [Test_I2C_Compass_Mag_Acc\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



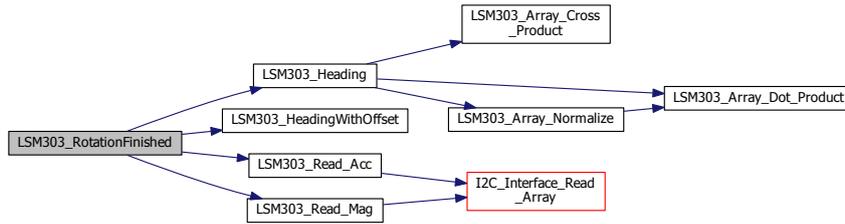
3.56.2.7 uint8_t LSM303_RotationFinished (void)

Definition at line 393 of file [LSM303.c](#).

References [LSM303_Heading\(\)](#), [LSM303_HeadingWithOffset\(\)](#), [LSM303_Read_Acc\(\)](#), and [LSM303_Read_Mag\(\)](#).

Referenced by [State_Machine_Update\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



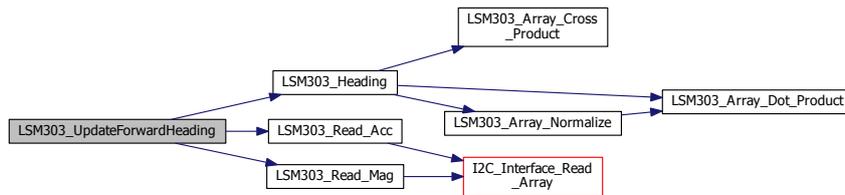
3.56.2.8 void LSM303_UpdateForwardHeading (void)

Definition at line 376 of file LSM303.c.

References [LSM303_Heading\(\)](#), [LSM303_Read_Acc\(\)](#), and [LSM303_Read_Mag\(\)](#).

Referenced by [State_Machine_State_2\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.57 LSM303.h

00001 /**

```

00002 * @file LSM303.h
00003 * @author Nicholas Sikkema
00004 * @version Revision: 1.0
00005 * @date Last Updated: 1/22/2015\n Created: 10/30/2014 4:42:22 PM
00006 * @brief This file is code to control the LSM303.
00007 */
00008
00009 #ifndef LSM303_H_
00010 #define LSM303_H_
00011
00012 #include <stdint.h>
00013
00014 void LSM303_Initialize();
00015 void LSM303_Read_Mag(int16_t *Mag);
00016 void LSM303_Read_Acc(int16_t *Acc);
00017 float LSM303_Heading(int16_t *Acc, int16_t *Mag);
00018 void LSM303_UpdateForwardHeading(void);
00019 float LSM303_HeadingWithOffset(float heading);
00020 uint8_t LSM303_RotationFinished(void);
00021 uint8_t LSM303_IsStablized(void);
00022
00023 #endif /* LSM303_H_ */

```

3.58 Subsystem/Light/CLS15.c File Reference

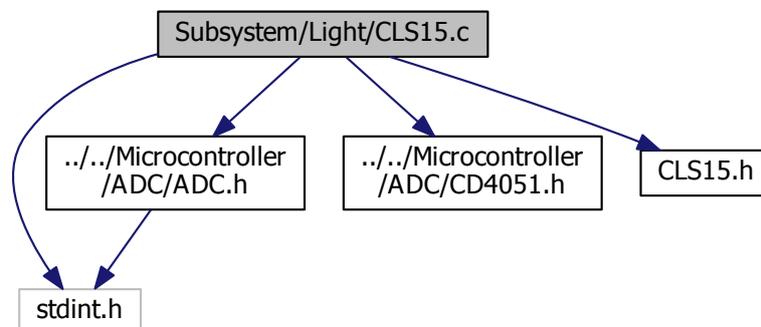
This file is code for the photodiode readings.

```

#include <stdint.h>
#include "../Microcontroller/ADC/ADC.h"
#include "../Microcontroller/ADC/CD4051.h"
#include "CLS15.h"

```

Include dependency graph for CLS15.c:



Functions

- float [CLS15_Ratio_Calculation](#) (uint16_t Front_Value, uint16_t Rear_Value)
Determines which Photodiode is reading higher and creates a ratio based on that knowledge.
- void [CLS15_Update_Photodiode_Value_Front](#) (void)
Update the front photodiode value.
- void [CLS15_Update_Photodiode_Value_Right](#) (void)
Update the right photodiode values.
- void [CLS15_Update_Photodiode_Value_Left](#) (void)
Update the left photodiode values.
- void [CLS15_Update_Photodiode_Values](#) (uint8_t Current_LED)

- Update all of the photodiode values.*
- uint16_t [CLS15_Retrieve_Photodiode_ADC_Front](#) (void)
Retrieve the ADC value for the front photodiode.
 - uint16_t [CLS15_Retrieve_Photodiode_ADC_Right_Front](#) (void)
Retrieve the ADC value for the right front photodiode.
 - uint16_t [CLS15_Retrieve_Photodiode_ADC_Right_Rear](#) (void)
Retrieve the ADC value for the right rear photodiode.
 - uint16_t [CLS15_Retrieve_Photodiode_ADC_Left_Front](#) (void)
Retrieve the ADC value for the left front photodiode.
 - uint16_t [CLS15_Retrieve_Photodiode_ADC_Left_Rear](#) (void)
Retrieve the ADC value for the left rear photodiode.
 - float [CLS15_Retrieve_Photodiode_Ratio_Right](#) (void)
Retrieve the ratio for the right photodiodes.
 - float [CLS15_Retrieve_Photodiode_Ratio_Left](#) (void)
Retrieve the ratio for the left photodiodes.
 - void [CLS15_Update_Photodiode_Offset_Front](#) (void)
 - void [CLS15_Update_Photodiode_Offset_Right_Front](#) (void)
 - void [CLS15_Update_Photodiode_Offset_Right_Rear](#) (void)
 - void [CLS15_Update_Photodiode_Offset_Left_Front](#) (void)
 - void [CLS15_Update_Photodiode_Offset_Left_Rear](#) (void)
 - void [CLS15_Update_Photodiode_Offset](#) (void)
Update the photodiode offsets.

3.58.1 Detailed Description

This file is code for the photodiode readings.

Author

Nicholas Sikkema

Version

Revision: 2.0

Date

Last Updated: 1/27/2015
Created: 12/2/2014 4:32:25 PM

Definition in file [CLS15.c](#).

3.58.2 Function Documentation

3.58.2.1 float [CLS15_Ratio_Calculation](#) (uint16_t *Front_Value*, uint16_t *Rear_Value*) `[inline]`

Determines which Photodiode is reading higher and creates a ratio based on that knowledge.

Parameters

<i>Front_Value</i>	The current value for the front (left or right) photodiode.
<i>Rear_Value</i>	The current value for the rear (left or right) photodiode.

Returns

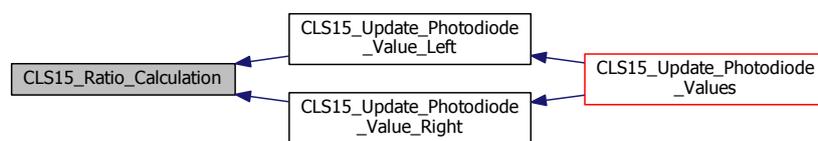
The ratio between the front and rear photodiode values. If > 1 then it is in the front, otherwise it is in the rear.

Return the ratio of the front value to the rear value.

Definition at line 36 of file [CLS15.c](#).

Referenced by [CLS15_Update_Photodiode_Value_Left\(\)](#), and [CLS15_Update_Photodiode_Value_Right\(\)](#).

Here is the caller graph for this function:



3.58.2.2 uint16_t CLS15_Retrieve_Photodiode_ADC_Front (void)

Retrieve the ADC value for the front photodiode.

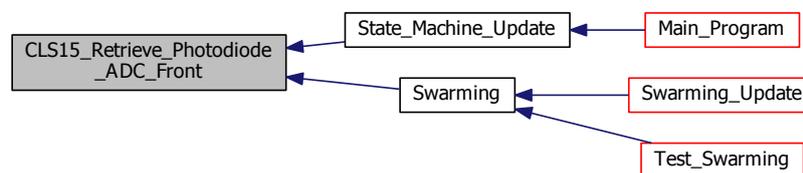
Returns

The ADC value for the front photodiode.

Definition at line 106 of file [CLS15.c](#).

Referenced by [State_Machine_Update\(\)](#), and [Swarming\(\)](#).

Here is the caller graph for this function:



3.58.2.3 uint16_t CLS15_Retrieve_Photodiode_ADC_Left_Front (void)

Retrieve the ADC value for the left front photodiode.

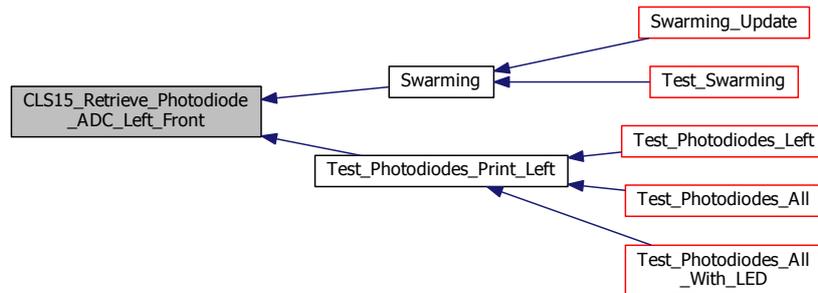
Returns

The ADC value for the left front photodiode.

Definition at line 133 of file [CLS15.c](#).

Referenced by [Swarming\(\)](#), and [Test_Photodiodes_Print_Left\(\)](#).

Here is the caller graph for this function:

**3.58.2.4 uint16_t CLS15_Retrieve_Photodiode_ADC_Left_Rear (void)**

Retrieve the ADC value for the left rear photodiode.

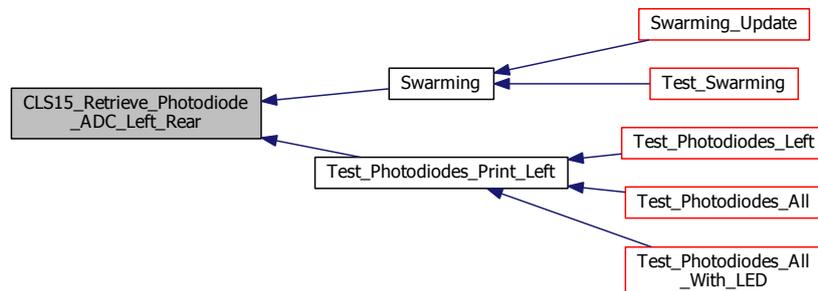
Returns

The ADC value for the left rear photodiode.

Definition at line 142 of file [CLS15.c](#).

Referenced by [Swarming\(\)](#), and [Test_Photodiodes_Print_Left\(\)](#).

Here is the caller graph for this function:

**3.58.2.5 uint16_t CLS15_Retrieve_Photodiode_ADC_Right_Front (void)**

Retrieve the ADC value for the right front photodiode.

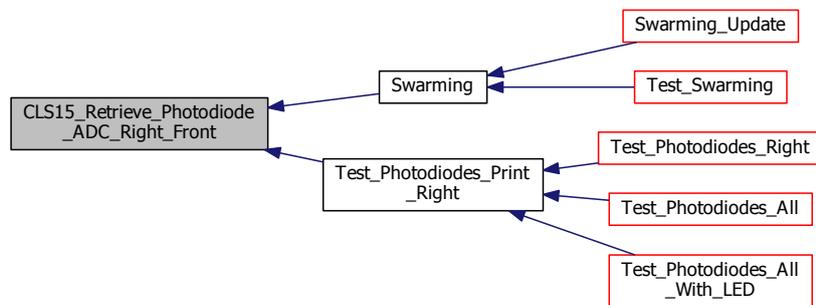
Returns

The ADC value for the right front photodiode.

Definition at line 115 of file [CLS15.c](#).

Referenced by [Swarming\(\)](#), and [Test_Photodiodes_Print_Right\(\)](#).

Here is the caller graph for this function:

**3.58.2.6 uint16_t CLS15_Retrieve_Photodiode_ADC_Right_Rear (void)**

Retrieve the ADC value for the right rear photodiode.

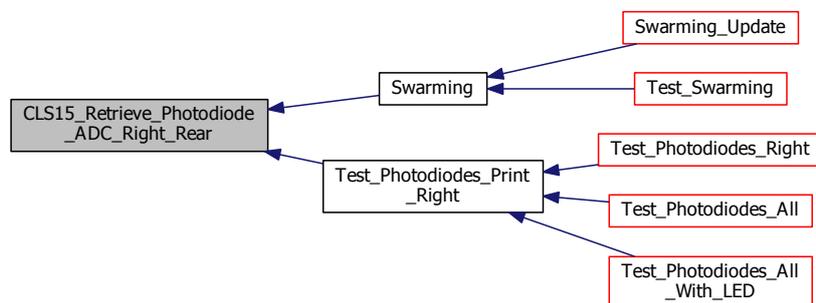
Returns

The ADC value for the right rear photodiode.

Definition at line 124 of file [CLS15.c](#).

Referenced by [Swarming\(\)](#), and [Test_Photodiodes_Print_Right\(\)](#).

Here is the caller graph for this function:

**3.58.2.7 float CLS15_Retrieve_Photodiode_Ratio_Left (void)**

Retrieve the ratio for the left photodiodes.

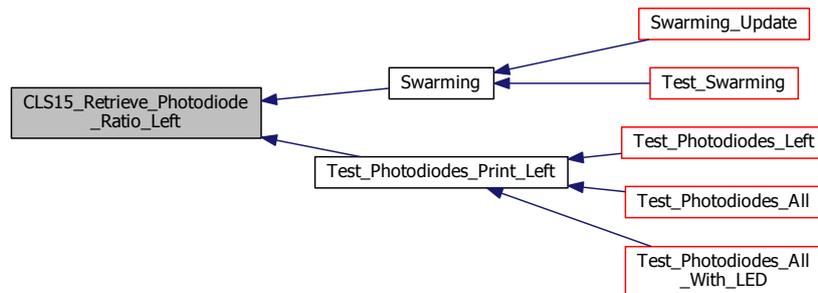
Returns

The ratio for the left photodiodes.

Definition at line 160 of file [CLS15.c](#).

Referenced by [Swarming\(\)](#), and [Test_Photodiodes_Print_Left\(\)](#).

Here is the caller graph for this function:



3.58.2.8 float CLS15_Retrieve_Photodiode_Ratio_Right (void)

Retrieve the ratio for the right photodiodes.

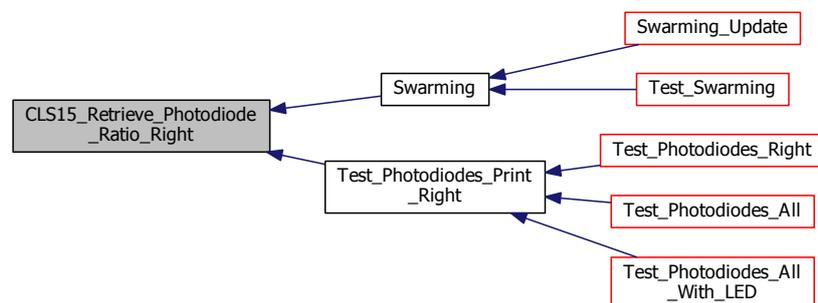
Returns

The ratio for the right photodiodes.

Definition at line 151 of file [CLS15.c](#).

Referenced by [Swarming\(\)](#), and [Test_Photodiodes_Print_Right\(\)](#).

Here is the caller graph for this function:



3.58.2.9 void CLS15_Update_Photodiode_Offset (void)

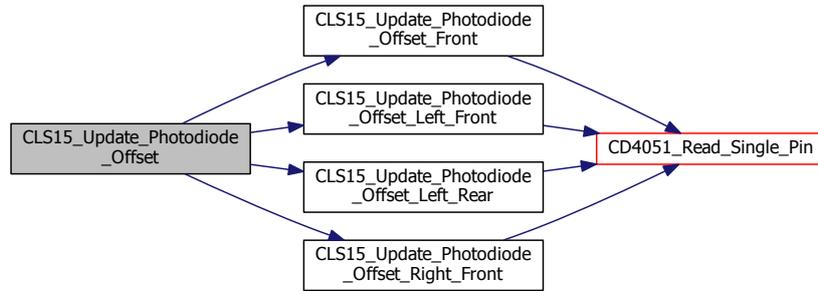
Update the photodiode offsets.

Definition at line 213 of file CLS15.c.

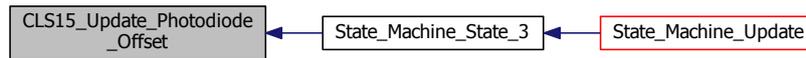
References [CLS15_Update_Photodiode_Offset_Front\(\)](#), [CLS15_Update_Photodiode_Offset_Left_Front\(\)](#), [CLS15_Update_Photodiode_Offset_Left_Rear\(\)](#), and [CLS15_Update_Photodiode_Offset_Right_Front\(\)](#).

Referenced by [State_Machine_State_3\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



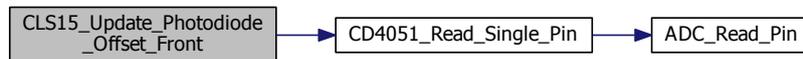
3.58.2.10 void CLS15_Update_Photodiode_Offset_Front (void)

Definition at line 165 of file CLS15.c.

References [CD4051_Read_Single_Pin\(\)](#), and [Photodiode_front](#).

Referenced by [CLS15_Update_Photodiode_Offset\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



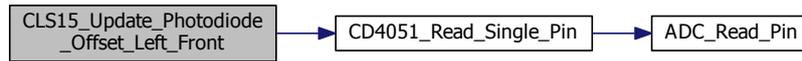
3.58.2.11 void CLS15_Update_Photodiode_Offset_Left_Front (void)

Definition at line 192 of file [CLS15.c](#).

References [CD4051_Read_Single_Pin\(\)](#), and [Photodiode_leftfront](#).

Referenced by [CLS15_Update_Photodiode_Offset\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



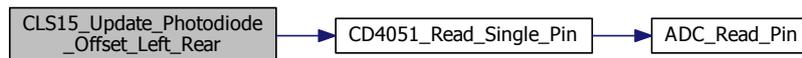
3.58.2.12 void CLS15_Update_Photodiode_Offset_Left_Rear (void)

Definition at line 201 of file [CLS15.c](#).

References [CD4051_Read_Single_Pin\(\)](#), and [Photodiode_leftrear](#).

Referenced by [CLS15_Update_Photodiode_Offset\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



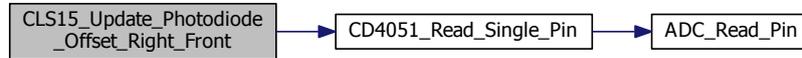
3.58.2.13 void CLS15_Update_Photodiode_Offset_Right_Front (void)

Definition at line 174 of file [CLS15.c](#).

References [CD4051_Read_Single_Pin\(\)](#), and [Photodiode_rightfront](#).

Referenced by [CLS15_Update_Photodiode_Offset\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

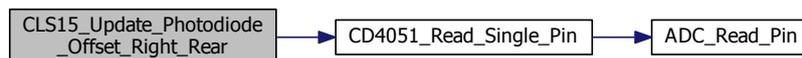


3.58.2.14 void CLS15_Update_Photodiode_Offset_Right_Rear (void)

Definition at line 183 of file [CLS15.c](#).

References [CD4051_Read_Single_Pin\(\)](#), and [Photodiode_rightrear](#).

Here is the call graph for this function:



3.58.2.15 void CLS15_Update_Photodiode_Value_Front (void)

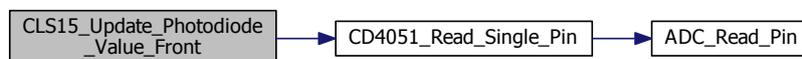
Update the front photodiode value.

Definition at line 45 of file [CLS15.c](#).

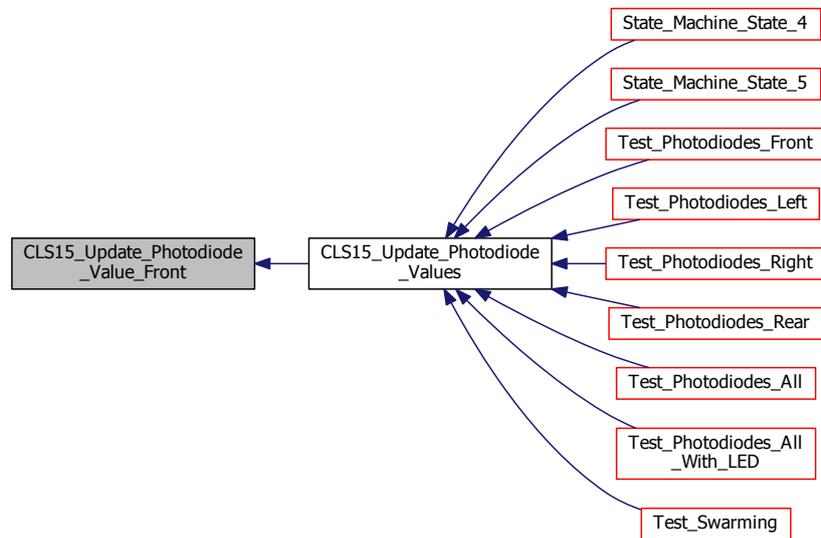
References [CD4051_Read_Single_Pin\(\)](#), and [Photodiode_front](#).

Referenced by [CLS15_Update_Photodiode_Values\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.58.2.16 void CLS15_Update_Photodiode_Value_Left (void)

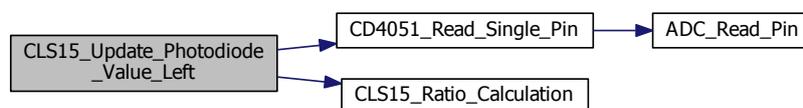
Update the left photodiode values.

Definition at line 63 of file [CLS15.c](#).

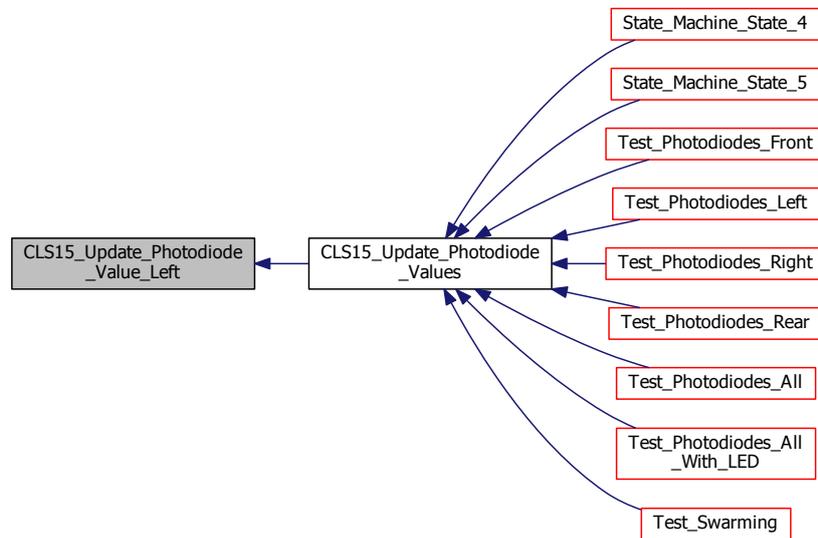
References [CD4051_Read_Single_Pin\(\)](#), [CLS15_Ratio_Calculation\(\)](#), [Photodiode_Lefffront](#), and [Photodiode_Leffrear](#).

Referenced by [CLS15_Update_Photodiode_Values\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.58.2.17 void CLS15_Update_Photodiode_Value_Right (void)

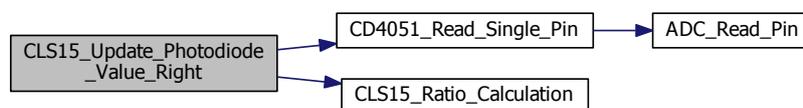
Update the right photodiode values.

Definition at line 53 of file CLS15.c.

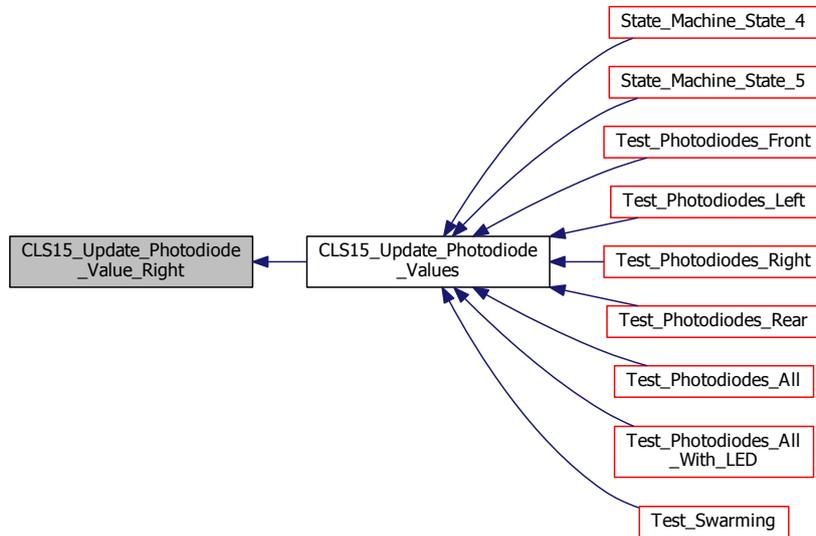
References [CD4051_Read_Single_Pin\(\)](#), [CLS15_Ratio_Calculation\(\)](#), [Photodiode_rightfront](#), and [Photodiode_rightrear](#).

Referenced by [CLS15_Update_Photodiode_Values\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.58.2.18 void CLS15_Update_Photodiode_Values (uint8_t Current_LED)

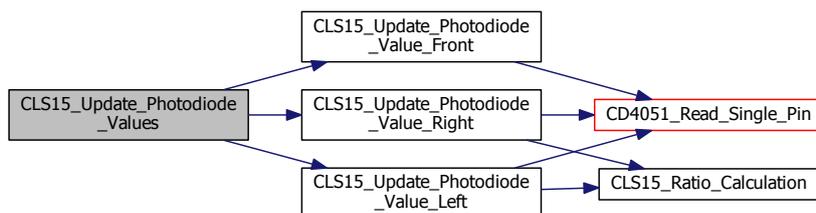
Update all of the photodiode values.

Definition at line 73 of file CLS15.c.

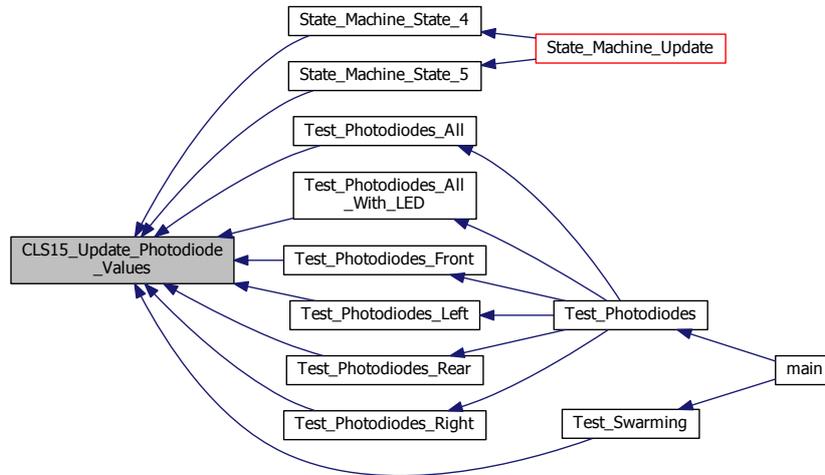
References CLS15_Update_Photodiode_Value_Front(), CLS15_Update_Photodiode_Value_Left(), and CLS15_Update_Photodiode_Value_Right().

Referenced by State_Machine_State_4(), State_Machine_State_5(), Test_Photodiodes_All(), Test_Photodiodes_All_With_LED(), Test_Photodiodes_Front(), Test_Photodiodes_Left(), Test_Photodiodes_Rear(), Test_Photodiodes_Right(), and Test_Swarming().

Here is the call graph for this function:



Here is the caller graph for this function:



3.59 CLS15.c

```

00001 /**
00002  * @file CLS15.c
00003  * @author Nicholas Sikkema
00004  * @version Revision: 2.0
00005  * @date Last Updated: 1/27/2015\n Created: 12/2/2014 4:32:25 PM
00006  * @brief This file is code for the photodiode readings.
00007  */
00008
00009 #include <stdint.h>
00010 #include "../Microcontroller/ADC/ADC.h"
00011 #include "../Microcontroller/ADC/CD4051.h"
00012 #include "CLS15.h"
00013
00014 /* Static variables to hold the photodiode values. */
00015 static uint16_t Photodiode_Value_Front = 0;
00016 static uint16_t Photodiode_Value_Right_Front = 0;
00017 static uint16_t Photodiode_Value_Right_Rear = 0;
00018 static uint16_t Photodiode_Value_Left_Front = 0;
00019 static uint16_t Photodiode_Value_Left_Rear = 0;
00020 /* Static variables to hold the photodiode offsets. */
00021 static uint16_t Photodiode_Value_Front_Offset = 5;
00022 static uint16_t Photodiode_Value_Right_Front_Offset = 5;
00023 static uint16_t Photodiode_Value_Right_Rear_Offset = 5;
00024 static uint16_t Photodiode_Value_Left_Front_Offset = 5;
00025 static uint16_t Photodiode_Value_Left_Rear_Offset = 5;
00026 /* Static variables to hold the ratios for the left and right sides. */
00027 static float left_ratio = 0;
00028 static float right_ratio = 0;
00029
00030 /**
00031  * @brief Determines which Photodiode is reading higher and creates a ratio based on that knowledge.
00032  * @param Front_Value The current value for the front (left or right) photodiode.
00033  * @param Rear_Value The current value for the rear (left or right) photodiode.
00034  * @return The ratio between the front and rear photodiode values. If > 1 then it is in the front,
00035  * otherwise it is in the rear.
00036  */
00037 inline float CLS15_Ratio_Calculation(uint16_t Front_Value, uint16_t Rear_Value)
00038 {
00039     /** Return the ratio of the front value to the rear value. */
00040     return((float)Front_Value/(float)Rear_Value);
00041 }
00042 /**
00043  * @brief Update the front photodiode value.
00044  */
00045 void CLS15_Update_Photodiode_Value_Front(void)
00046 {
00047     Photodiode_Value_Front = CD4051_Read_Single_Pin(
00048         Photodiode_front)-Photodiode_Value_Front_Offset;

```

```

00048 }
00049
00050 /**
00051  * @brief Update the right photodiode values.
00052  */
00053 void CLS15_Update_Photodiode_Value_Right(void)
00054 {
00055     Photodiode_Value_Right_Front = CD4051_Read_Single_Pin(
Photodiode_rightfront)-Photodiode_Value_Right_Front_Offset;
00056     Photodiode_Value_Right_Rear = CD4051_Read_Single_Pin(
Photodiode_rightrear)-Photodiode_Value_Right_Rear_Offset;
00057     right_ratio = CLS15_Ratio_Calculation(Photodiode_Value_Right_Front,
Photodiode_Value_Right_Rear);
00058 }
00059
00060 /**
00061  * @brief Update the left photodiode values.
00062  */
00063 void CLS15_Update_Photodiode_Value_Left(void)
00064 {
00065     Photodiode_Value_Left_Front = CD4051_Read_Single_Pin(
Photodiode_leftfront)-Photodiode_Value_Left_Front_Offset;
00066     Photodiode_Value_Left_Rear = CD4051_Read_Single_Pin(
Photodiode_leftrear)-Photodiode_Value_Left_Rear_Offset;
00067     left_ratio = CLS15_Ratio_Calculation(Photodiode_Value_Left_Front,
Photodiode_Value_Left_Rear);
00068 }
00069
00070 /**
00071  * @brief Update all of the photodiode values.
00072  */
00073 void CLS15_Update_Photodiode_Values(uint8_t Current_LED)
00074 {
00075     if(Current_LED == 1)
00076     {
00077         CLS15_Update_Photodiode_Value_Right();
00078         CLS15_Update_Photodiode_Value_Left();
00079     }
00080     else if(Current_LED == 2)
00081     {
00082         CLS15_Update_Photodiode_Value_Front();
00083         CLS15_Update_Photodiode_Value_Left();
00084     }
00085     else if(Current_LED == 4)
00086     {
00087         CLS15_Update_Photodiode_Value_Front();
00088         CLS15_Update_Photodiode_Value_Right();
00089     }
00090     else if(Current_LED == 8)
00091     {
00092         CLS15_Update_Photodiode_Value_Front();
00093         CLS15_Update_Photodiode_Value_Right();
00094         CLS15_Update_Photodiode_Value_Left();
00095     }
00096     else if (Current_LED == 16)
00097     {
00098         CLS15_Update_Photodiode_Value_Front();
00099     }
00100 }
00101
00102 /**
00103  * @brief Retrieve the ADC value for the front photodiode.
00104  * @return The ADC value for the front photodiode.
00105  */
00106 uint16_t CLS15_Retrieve_Photodiode_ADC_Front(void)
00107 {
00108     return(Photodiode_Value_Front);
00109 }
00110
00111 /**
00112  * @brief Retrieve the ADC value for the right front photodiode.
00113  * @return The ADC value for the right front photodiode.
00114  */
00115 uint16_t CLS15_Retrieve_Photodiode_ADC_Right_Front(void)
00116 {
00117     return(Photodiode_Value_Right_Front);
00118 }
00119
00120 /**
00121  * @brief Retrieve the ADC value for the right rear photodiode.
00122  * @return The ADC value for the right rear photodiode.
00123  */
00124 uint16_t CLS15_Retrieve_Photodiode_ADC_Right_Rear(void)
00125 {
00126     return(Photodiode_Value_Right_Rear);
00127 }
00128

```

```
00129 /**
00130  * @brief Retrieve the ADC value for the left front photodiode.
00131  * @return The ADC value for the left front photodiode.
00132  */
00133 uint16_t CLS15_Retrieve_Photodiode_ADC_Left_Front(void)
00134 {
00135     return(Photodiode_Value_Left_Front);
00136 }
00137
00138 /**
00139  * @brief Retrieve the ADC value for the left rear photodiode.
00140  * @return The ADC value for the left rear photodiode.
00141  */
00142 uint16_t CLS15_Retrieve_Photodiode_ADC_Left_Rear(void)
00143 {
00144     return(Photodiode_Value_Left_Rear);
00145 }
00146
00147 /**
00148  * @brief Retrieve the ratio for the right photodiodes.
00149  * @return The ratio for the right photodiodes.
00150  */
00151 float CLS15_Retrieve_Photodiode_Ratio_Right(void)
00152 {
00153     return(right_ratio);
00154 }
00155
00156 /**
00157  * @brief Retrieve the ratio for the left photodiodes.
00158  * @return The ratio for the left photodiodes.
00159  */
00160 float CLS15_Retrieve_Photodiode_Ratio_Left(void)
00161 {
00162     return(left_ratio);
00163 }
00164
00165 void CLS15_Update_Photodiode_Offset_Front(void)
00166 {
00167     uint16_t Temp_Value = CD4051_Read_Single_Pin(
Photodiode_front);
00168     if(Temp_Value<Photodiode_Value_Front_Offset)
00169     {
00170         Photodiode_Value_Front_Offset = Temp_Value;
00171     }
00172 }
00173
00174 void CLS15_Update_Photodiode_Offset_Right_Front(void)
00175 {
00176     uint16_t Temp_Value = CD4051_Read_Single_Pin(
Photodiode_rightfront);
00177     if(Temp_Value<Photodiode_Value_Front_Offset)
00178     {
00179         Photodiode_Value_Right_Front_Offset = Temp_Value;
00180     }
00181 }
00182
00183 void CLS15_Update_Photodiode_Offset_Right_Rear(void)
00184 {
00185     uint16_t Temp_Value = CD4051_Read_Single_Pin(
Photodiode_rightrear);
00186     if(Temp_Value<Photodiode_Value_Front_Offset)
00187     {
00188         Photodiode_Value_Right_Rear_Offset = Temp_Value;
00189     }
00190 }
00191
00192 void CLS15_Update_Photodiode_Offset_Left_Front(void)
00193 {
00194     uint16_t Temp_Value = CD4051_Read_Single_Pin(
Photodiode_leftfront);
00195     if(Temp_Value<Photodiode_Value_Front_Offset)
00196     {
00197         Photodiode_Value_Left_Front_Offset = Temp_Value;
00198     }
00199 }
00200
00201 void CLS15_Update_Photodiode_Offset_Left_Rear(void)
00202 {
00203     uint16_t Temp_Value = CD4051_Read_Single_Pin(
Photodiode_leftrear);
00204     if(Temp_Value<Photodiode_Value_Front_Offset)
00205     {
00206         Photodiode_Value_Left_Rear_Offset = Temp_Value;
00207     }
00208 }
00209
00210 /**
```

```

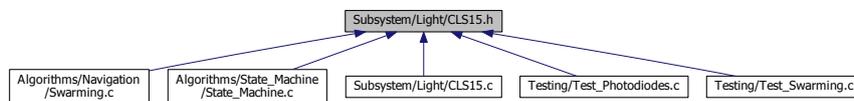
00211  * @brief Update the photodiode offsets.
00212  */
00213  void CLS15_Update_Photodiode_Offset(void)
00214  {
00215      /* Update the front photodiode offset. */
00216      CLS15_Update_Photodiode_Offset_Front();
00217      /* Update the front photodiode offset. */
00218      CLS15_Update_Photodiode_Offset_Left_Front();
00219      /* Update the front photodiode offset. */
00220      CLS15_Update_Photodiode_Offset_Left_Rear();
00221      /* Update the front photodiode offset. */
00222      CLS15_Update_Photodiode_Offset_Right_Front();
00223  }

```

3.60 Subsystem/Light/CLS15.h File Reference

This is the header file for the photodiode reading code.

This graph shows which files directly or indirectly include this file:



Functions

- void [CLS15_Update_Photodiode_Values](#) (uint8_t Current_LED)
Update all of the photodiode values.
- uint16_t [CLS15_Retrieve_Photodiode_ADC_Front](#) (void)
Retrieve the ADC value for the front photodiode.
- uint16_t [CLS15_Retrieve_Photodiode_ADC_Right_Front](#) (void)
Retrieve the ADC value for the right front photodiode.
- uint16_t [CLS15_Retrieve_Photodiode_ADC_Right_Rear](#) (void)
Retrieve the ADC value for the right rear photodiode.
- uint16_t [CLS15_Retrieve_Photodiode_ADC_Left_Front](#) (void)
Retrieve the ADC value for the left front photodiode.
- uint16_t [CLS15_Retrieve_Photodiode_ADC_Left_Rear](#) (void)
Retrieve the ADC value for the left rear photodiode.
- uint16_t [CLS15_Retrieve_Photodiode_ADC_Rear](#) (void)
- float [CLS15_Retrieve_Photodiode_Ratio_Right](#) (void)
Retrieve the ratio for the right photodiodes.
- float [CLS15_Retrieve_Photodiode_Ratio_Left](#) (void)
Retrieve the ratio for the left photodiodes.
- void [CLS15_Update_Photodiode_Offset](#) (void)
Update the photodiode offsets.

3.60.1 Detailed Description

This is the header file for the photodiode reading code.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 1/27/2015

Created: 12/2/2014 4:32:25 PM

Definition in file [CLS15.h](#).**3.60.2 Function Documentation****3.60.2.1 uint16_t CLS15_Retrieve_Photodiode_ADC_Front (void)**

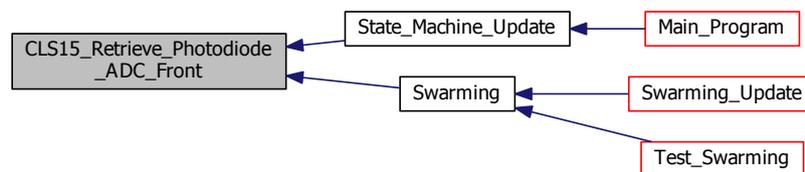
Retrieve the ADC value for the front photodiode.

Returns

The ADC value for the front photodiode.

Definition at line 106 of file [CLS15.c](#).Referenced by [State_Machine_Update\(\)](#), and [Swarming\(\)](#).

Here is the caller graph for this function:

**3.60.2.2 uint16_t CLS15_Retrieve_Photodiode_ADC_Left_Front (void)**

Retrieve the ADC value for the left front photodiode.

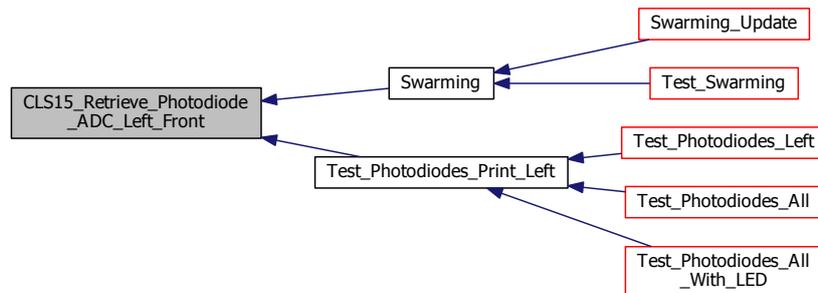
Returns

The ADC value for the left front photodiode.

Definition at line 133 of file [CLS15.c](#).

Referenced by [Swarming\(\)](#), and [Test_Photodiodes_Print_Left\(\)](#).

Here is the caller graph for this function:

**3.60.2.3 uint16_t CLS15_Retrieve_Photodiode_ADC_Left_Rear (void)**

Retrieve the ADC value for the left rear photodiode.

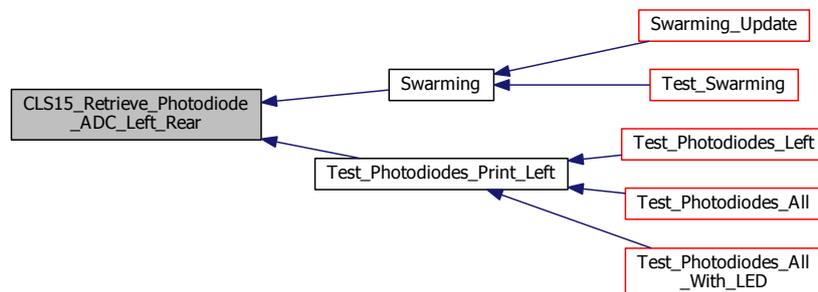
Returns

The ADC value for the left rear photodiode.

Definition at line 142 of file [CLS15.c](#).

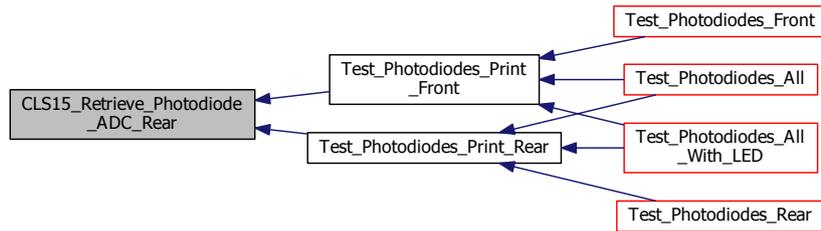
Referenced by [Swarming\(\)](#), and [Test_Photodiodes_Print_Left\(\)](#).

Here is the caller graph for this function:

**3.60.2.4 uint16_t CLS15_Retrieve_Photodiode_ADC_Rear (void)**

Referenced by [Test_Photodiodes_Print_Front\(\)](#), and [Test_Photodiodes_Print_Rear\(\)](#).

Here is the caller graph for this function:



3.60.2.5 uint16_t CLS15_Retrieve_Photodiode_ADC_Right_Front (void)

Retrieve the ADC value for the right front photodiode.

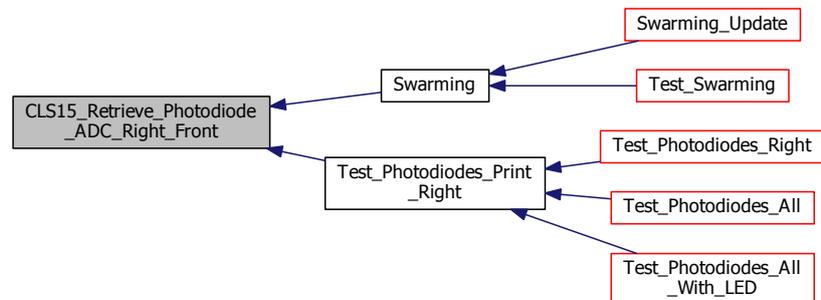
Returns

The ADC value for the right front photodiode.

Definition at line 115 of file [CLS15.c](#).

Referenced by [Swarming\(\)](#), and [Test_Photodiodes_Print_Right\(\)](#).

Here is the caller graph for this function:



3.60.2.6 uint16_t CLS15_Retrieve_Photodiode_ADC_Right_Rear (void)

Retrieve the ADC value for the right rear photodiode.

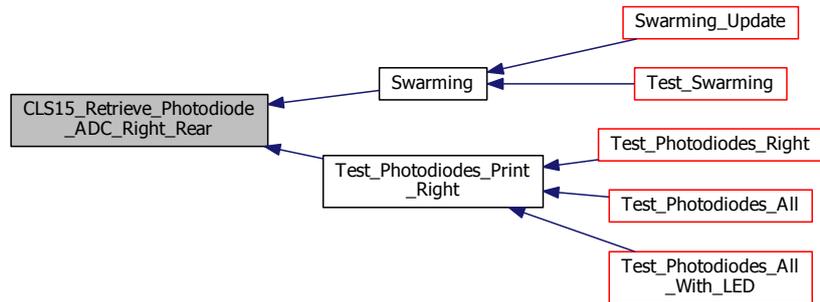
Returns

The ADC value for the right rear photodiode.

Definition at line 124 of file [CLS15.c](#).

Referenced by [Swarming\(\)](#), and [Test_Photodiodes_Print_Right\(\)](#).

Here is the caller graph for this function:



3.60.2.7 float CLS15_Retrieve_Photosodiode_Ratio_Left (void)

Retrieve the ratio for the left photosodiodes.

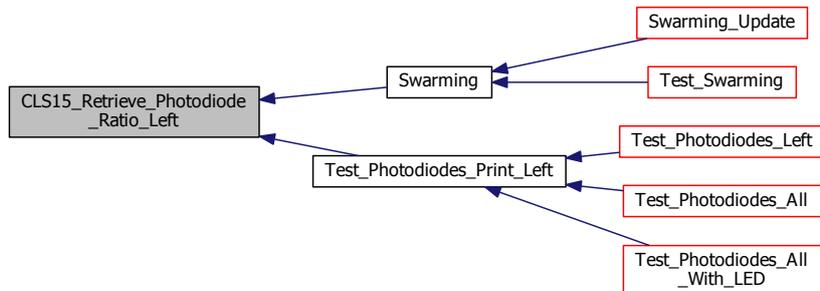
Returns

The ratio for the left photosodiodes.

Definition at line 160 of file [CLS15.c](#).

Referenced by [Swarming\(\)](#), and [Test_Photosodiodes_Print_Left\(\)](#).

Here is the caller graph for this function:



3.60.2.8 float CLS15_Retrieve_Photosodiode_Ratio_Right (void)

Retrieve the ratio for the right photosodiodes.

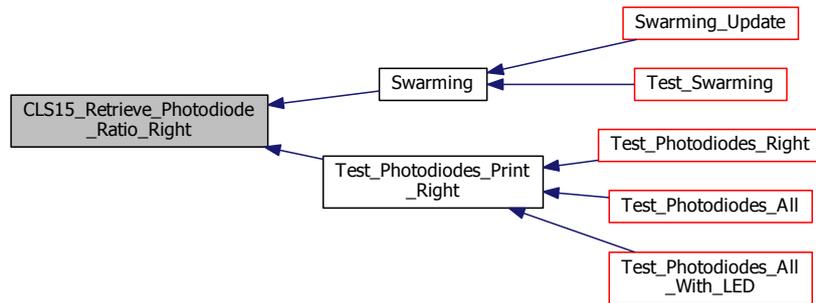
Returns

The ratio for the right photosodiodes.

Definition at line 151 of file [CLS15.c](#).

Referenced by [Swarming\(\)](#), and [Test_Photosodiodes_Print_Right\(\)](#).

Here is the caller graph for this function:



3.60.2.9 void CLS15_Update_Photodiode_Offset (void)

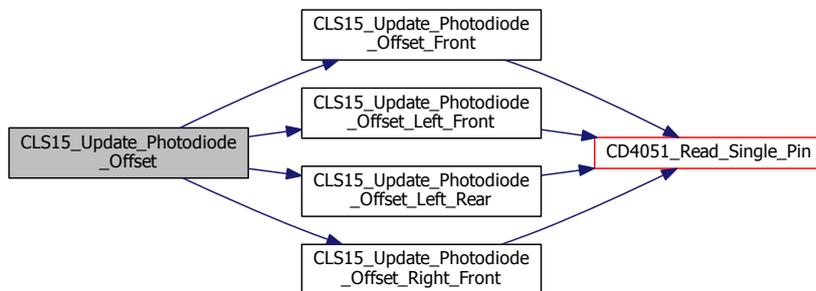
Update the photodiode offsets.

Definition at line 213 of file CLS15.c.

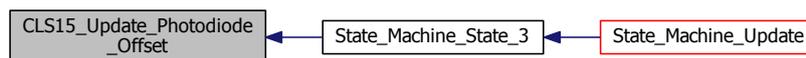
References [CLS15_Update_Photodiode_Offset_Front\(\)](#), [CLS15_Update_Photodiode_Offset_Left_Front\(\)](#), [CLS15_Update_Photodiode_Offset_Left_Rear\(\)](#), and [CLS15_Update_Photodiode_Offset_Right_Front\(\)](#).

Referenced by [State_Machine_State_3\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.60.2.10 void CLS15_Update_Photodiode_Values (uint8_t Current_LED)

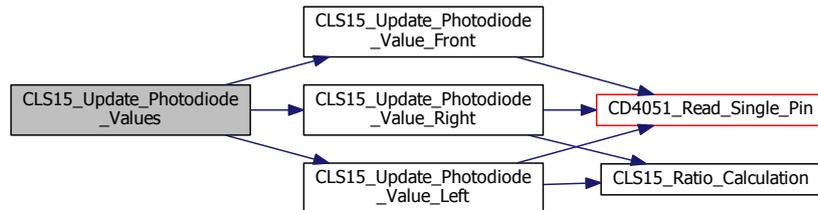
Update all of the photodiode values.

Definition at line 73 of file CLS15.c.

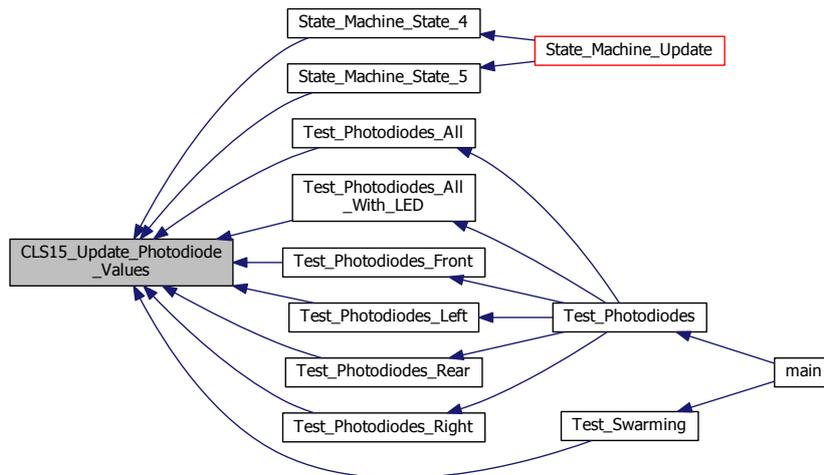
References [CLS15_Update_Photodiode_Value_Front\(\)](#), [CLS15_Update_Photodiode_Value_Left\(\)](#), and [CLS15_Update_Photodiode_Value_Right\(\)](#).

Referenced by [State_Machine_State_4\(\)](#), [State_Machine_State_5\(\)](#), [Test_Photodiodes_All\(\)](#), [Test_Photodiodes_All_With_LED\(\)](#), [Test_Photodiodes_Front\(\)](#), [Test_Photodiodes_Left\(\)](#), [Test_Photodiodes_Rear\(\)](#), [Test_Photodiodes_Right\(\)](#), and [Test_Swarming\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.61 CLS15.h

```

00001 /**
00002  * @file CLS15.h
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 1/27/2015\n Created: 12/2/2014 4:32:25 PM
00006  * @brief This is the header file for the photodiode reading code.
00007  */
00008
00009 #ifndef CLS15_H_
00010 #define CLS15_H_
00011
00012 void CLS15_Update_Photodiode_Values(uint8_t Current_LED);
00013 uint16_t CLS15_Retrieve_Photodiode_ADC_Front(void);
00014 uint16_t CLS15_Retrieve_Photodiode_ADC_Right_Front(void);
00015 uint16_t CLS15_Retrieve_Photodiode_ADC_Right_Rear(void);
00016 uint16_t CLS15_Retrieve_Photodiode_ADC_Left_Front(void);
  
```

```

00017 uint16_t CLS15_Retrieve_Photodiode_ADC_Left_Rear(void);
00018 uint16_t CLS15_Retrieve_Photodiode_ADC_Rear(void);
00019 float CLS15_Retrieve_Photodiode_Ratio_Right(void);
00020 float CLS15_Retrieve_Photodiode_Ratio_Left(void);
00021 void CLS15_Update_Photodiode_Offset(void);
00022
00023 #endif /* CLS15_H_ */

```

3.62 Subsystem/Light/XPEBBL.c File Reference

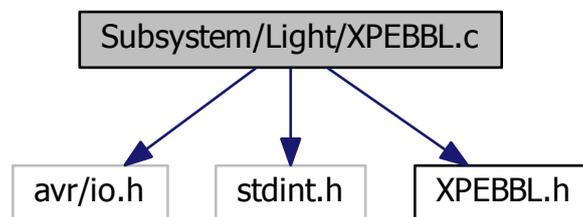
This file is code to control the 3 Watt LEDs.

```

#include <avr/io.h>
#include <stdint.h>
#include "XPEBBL.h"

```

Include dependency graph for XPEBBL.c:



Macros

- #define `Cycle_State` 3
Max number of states for the change LED function to be in.
- #define `Front_LED_Bit` 5
Bit position of the front LED.

Functions

- void `XPEBBL_Initialize` (void)
Initialize the ports used for the LEDs.
- void `XPEBBL_Set_Pin` (unsigned char LED_Pin)
Updates which led is on.
- uint8_t `XPEBBL_Toggle_Front_LED` ()
This function toggles the front LED.
- uint8_t `XPEBLL_Change_LED` (void)
Changes which LED that is currently on.

3.62.1 Detailed Description

This file is code to control the 3 Watt LEDs.

Author

Nicholas Sikkema

Version

Revision: 2.0

Date

Last Updated: 1/22/2015

Created: 12/2/2014 3:07:03 PM

Definition in file [XPEBBL.c](#).**3.62.2 Macro Definition Documentation****3.62.2.1 #define Cycle_State 3**

Max number of states for the change LED function to be in.

Definition at line 10 of file [XPEBBL.c](#).Referenced by [XPEBLL_Change_LED\(\)](#).**3.62.2.2 #define Front_LED_Bit 5**

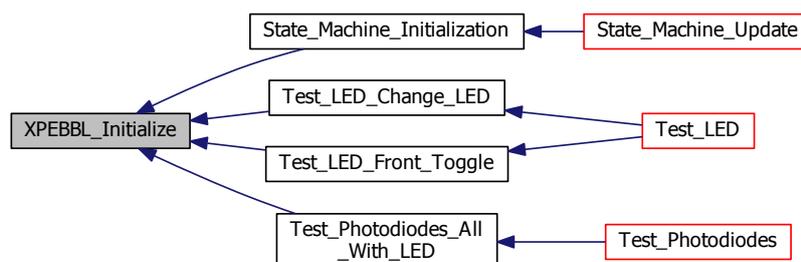
Bit position of the front LED.

Definition at line 12 of file [XPEBBL.c](#).Referenced by [XPEBBL_Toggle_Front_LED\(\)](#).**3.62.3 Function Documentation****3.62.3.1 void XPEBBL_Initialize (void)**

Initialize the ports used for the LEDs.

Definition at line 21 of file [XPEBBL.c](#).Referenced by [State_Machine_Initialization\(\)](#), [Test_LED_Change_LED\(\)](#), [Test_LED_Front_Toggle\(\)](#), and [Test_Photodiodes_All_With_LED\(\)](#).

Here is the caller graph for this function:



3.62.3.2 void XPEBBL_Set_Pin (unsigned char *LED_Pin*)

Updates which led is on.

Parameters

<i>LED_Pin</i>	The pin to be set.
----------------	--------------------

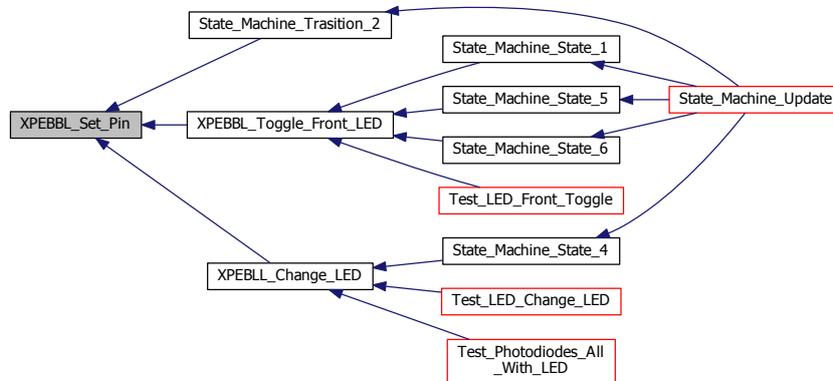
Warning

This function does not check the input.

Definition at line 32 of file [XPEBBL.c](#).

Referenced by [State_Machine_Trasition_2\(\)](#), [XPEBBL_Toggle_Front_LED\(\)](#), and [XPEBLL_Change_LED\(\)](#).

Here is the caller graph for this function:



3.62.3.3 uint8_t XPEBBL_Toggle_Front_LED (void)

This function toggles the front LED.

Definition at line 43 of file [XPEBBL.c](#).

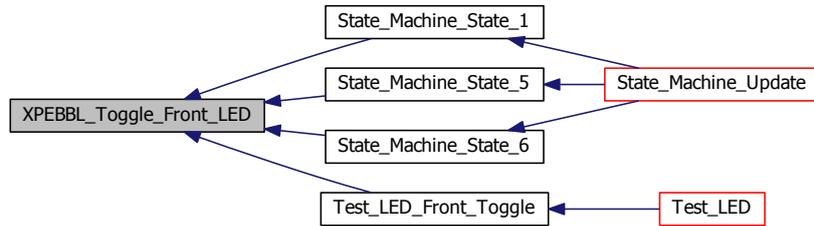
References [Front_LED_Bit](#), and [XPEBBL_Set_Pin\(\)](#).

Referenced by [State_Machine_State_1\(\)](#), [State_Machine_State_5\(\)](#), [State_Machine_State_6\(\)](#), and [Test_LED_Front_Toggle\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.62.3.4 uint8_t XPEBLL_Change_LED (void)

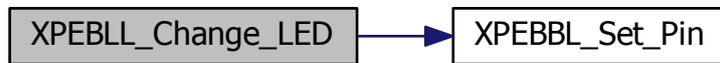
Changes which LED that is currently on.

Definition at line 70 of file XPEBBL.c.

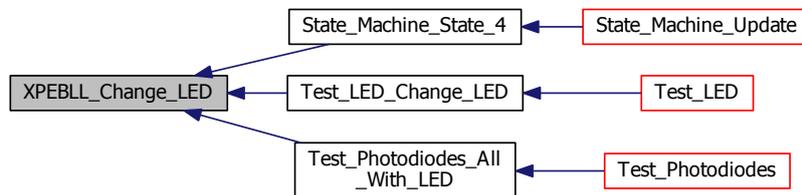
References [Cycle_State](#), and [XPEBBL_Set_Pin\(\)](#).

Referenced by [State_Machine_State_4\(\)](#), [Test_LED_Change_LED\(\)](#), and [Test_Photodiodes_All_With_LED\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.63 XPEBBL.c

```

00001 /**
00002  * @file XPEBBL.c
00003  * @author Nicholas Sikkema
00004  * @version Revision: 2.0
    
```

```

00005  * @date Last Updated: 1/22/2015\n Created: 12/2/2014 3:07:03 PM
00006  * @brief This file is code to control the 3 Watt LEDs.
00007  */
00008
00009  /** @brief Max number of states for the change LED function to be in. */
00010  #define Cycle_State 3
00011  /** @brief Bit position of the front LED */
00012  #define Front_LED_Bit 5
00013
00014  #include <avr/io.h>
00015  #include <stdint.h>
00016  #include "XPEBBL.h"
00017
00018  /**
00019   * @brief Initialize the ports used for the LEDs.
00020   */
00021  void XPEBBL_Initialize(void)
00022  {
00023      /* Initialize the Led Ports to be an output. */
00024      DDRB |= (1<<DDB5)|(1<<DDB4)|(1<<DDB3);
00025  }
00026
00027  /**
00028   * @brief Updates which led is on.
00029   * @param LED_Pin The pin to be set.
00030   * @warning This function does not check the input.
00031   */
00032  void XPEBBL_Set_Pin(unsigned char LED_Pin)
00033  {
00034      /* Clear the bits 3, 4, and 5. */
00035      PORTB &= 0xC7;
00036      /* Set the LED value in the Port. */
00037      PORTB |= LED_Pin;
00038  }
00039
00040  /**
00041   * @brief This function toggles the front LED.
00042   */
00043  uint8_t XPEBBL_Toggle_Front_LED()
00044  {
00045      /* Initialize the previous value variable. */
00046      static uint8_t Previous_Value = 0;
00047      /* Initialize the current value variable. */
00048      uint8_t Current_Value = 0;
00049      /* Check if the previous value is zero. */
00050      if(Previous_Value == 0)
00051      {
00052          /* Enable the bit for the front LED. */
00053          Current_Value = 1 << Front_LED_Bit;
00054      }
00055      else
00056      {
00057          /* Clear the bit for the front LED. */
00058          Current_Value = 0;
00059      }
00060      /* Set the LED value. */
00061      XPEBBL_Set_Pin(Current_Value);
00062      /* Update the previous value. */
00063      Previous_Value = Current_Value;
00064      return(~(Current_Value>0));
00065  }
00066
00067  /**
00068   * @brief Changes which LED that is currently on.
00069   */
00070  uint8_t XPEBLL_Change_LED(void)
00071  {
00072      /* Initialize the current pin variable. */
00073      static uint8_t Current_Pin = 0x08;
00074      /* Initialize the LED state variable. */
00075      static uint8_t LED_State = 0;
00076      /* Change LED State. */
00077      if(LED_State>Cycle_State)
00078      {
00079          /* Change the current LED that is on. */
00080          Current_Pin = Current_Pin << 1;
00081          /* Check to see if the current pin is in bounds. */
00082          if(Current_Pin > 0x20)
00083          {
00084              /* Reinitialize the current pin value. */
00085              Current_Pin = 0x08;
00086          }
00087          /* Reinitialize the state of the LED. */
00088          LED_State = 0;
00089          /* Set the LED value. */
00090          XPEBBL_Set_Pin(Current_Pin);
00091      }

```

```

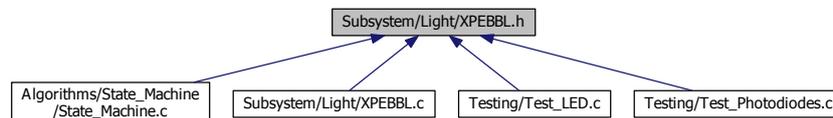
00092     /* Turn off State. */
00093     else if(LED_State == Cycle_State)
00094     {
00095         /* Turn off the LEDs. */
00096         XPEBBL_Set_Pin(0x00);
00097     }
00098     /* Stay on State. */
00099     else
00100     {
00101         /* Make sure that the LED value is what is wanted. */
00102         XPEBBL_Set_Pin(Current_Pin);
00103     }
00104     /* Increment the state of the LEDs.*/
00105     LED_State++;
00106     /* Return the current pin. */
00107     return(Current_Pin>>3);
00108 }

```

3.64 Subsystem/Light/XPEBBL.h File Reference

This file is code to control the 3 Watt LEDs.

This graph shows which files directly or indirectly include this file:



Functions

- void [XPEBBL_Initialize](#) (void)
Initialize the ports used for the LEDs.
- void [XPEBBL_Set_Pin](#) (unsigned char LED_Pin)
Updates which led is on.
- uint8_t [XPEBLL_Change_LED](#) (void)
Changes which LED that is currently on.
- uint8_t [XPEBBL_Toggle_Front_LED](#) (void)
This function toggles the front LED.

3.64.1 Detailed Description

This file is code to control the 3 Watt LEDs.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 1/22/2015

Created: 12/2/2014 3:07:26 PM

Definition in file [XPEBBL.h](#).

3.64.2 Function Documentation

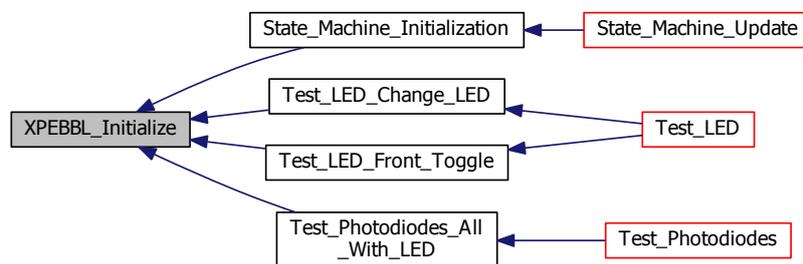
3.64.2.1 void XPEBBL_Initialize (void)

Initialize the ports used for the LEDs.

Definition at line 21 of file [XPEBBL.c](#).

Referenced by [State_Machine_Initialization\(\)](#), [Test_LED_Change_LED\(\)](#), [Test_LED_Front_Toggle\(\)](#), and [Test_Photodiodes_All_With_LED\(\)](#).

Here is the caller graph for this function:



3.64.2.2 void XPEBBL_Set_Pin (unsigned char LED_Pin)

Updates which led is on.

Parameters

<i>LED_Pin</i>	The pin to be set.
----------------	--------------------

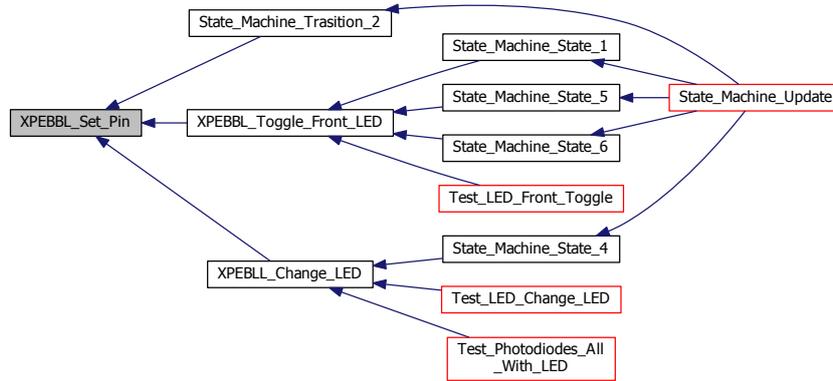
Warning

This function does not check the input.

Definition at line 32 of file [XPEBBL.c](#).

Referenced by [State_Machine_Trasition_2\(\)](#), [XPEBBL_Toggle_Front_LED\(\)](#), and [XPEBLL_Change_LED\(\)](#).

Here is the caller graph for this function:



3.64.2.3 uint8_t XPEBBL_Toggle_Front_LED (void)

This function toggles the front LED.

Definition at line 43 of file XPEBBL.c.

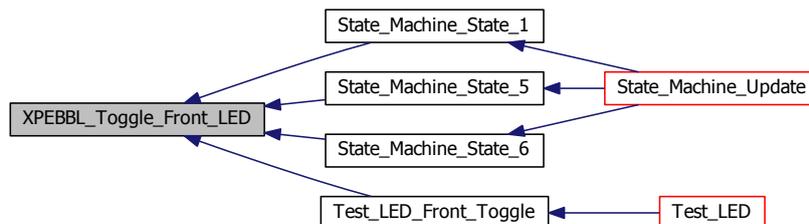
References [Front_LED_Bit](#), and [XPEBBL_Set_Pin\(\)](#).

Referenced by [State_Machine_State_1\(\)](#), [State_Machine_State_5\(\)](#), [State_Machine_State_6\(\)](#), and [Test_LED_Front_Toggle\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.64.2.4 uint8_t XPEBLL_Change_LED (void)

Changes which LED that is currently on.

Definition at line 70 of file [XPEBBL.c](#).

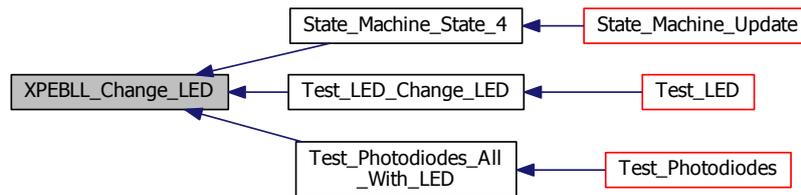
References [Cycle_State](#), and [XPEBBL_Set_Pin\(\)](#).

Referenced by [State_Machine_State_4\(\)](#), [Test_LED_Change_LED\(\)](#), and [Test_Photodiodes_All_With_LED\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.65 XPEBBL.h

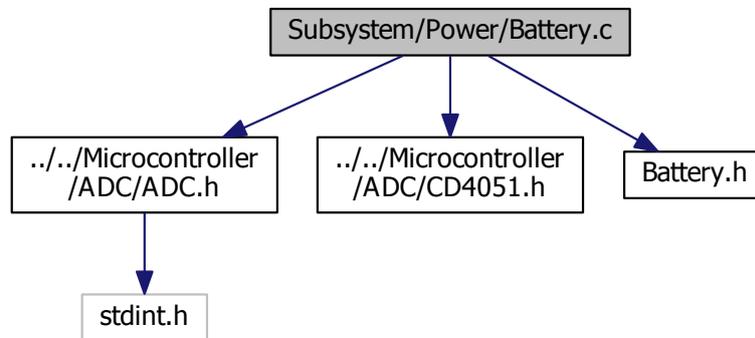
```

00001 /**
00002  * @file XPEBBL.h
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 1/22/2015\n Created: 12/2/2014 3:07:26 PM
00006  * @brief This file is code to control the 3 Watt LEDs.
00007  */
00008
00009 #ifndef XPEBBL_H_
00010 #define XPEBBL_H_
00011
00012 void XPEBBL_Initialize(void);
00013 void XPEBBL_Set_Pin(unsigned char LED_Pin);
00014 uint8_t XPEBLL_Change_LED(void);
00015 uint8_t XPEBBL_Toggle_Front_LED(void);
00016
00017 #endif /* XPEBBL_H_ */
  
```

3.66 Subsystem/Power/Battery.c File Reference

This file is code to read the battery voltage.

```
#include "../..//Microcontroller/ADC/ADC.h"
#include "../..//Microcontroller/ADC/CD4051.h"
#include "Battery.h"
Include dependency graph for Battery.c:
```



Macros

- `#define Battery_Safe_Voltage 3.45`
This is the lowest safe voltage value.

Functions

- `int Battery_Read_ADC ()`
This function is used to read the ADC port for the battery.
- `float Battery_Read_Voltage ()`
This function is used to read the battery voltage.
- `int Battery_Check_Status ()`
This function is used to check to see if the battery is below X volts.

3.66.1 Detailed Description

This file is code to read the battery voltage.

Author

Nicholas Sikkema

Version

Revision: 2.0

Date

Last Updated: 1/22/2015
Created: 11/13/2014 2:33:44 PM

Definition in file [Battery.c](#).

3.66.2 Macro Definition Documentation

3.66.2.1 #define Battery_Safe_Voltage 3.45

This is the lowest safe voltage value.

Definition at line 10 of file [Battery.c](#).

Referenced by [Battery_Check_Status\(\)](#).

3.66.3 Function Documentation

3.66.3.1 int Battery_Check_Status (void)

This function is used to check to see if the battery is below X volts.

Returns

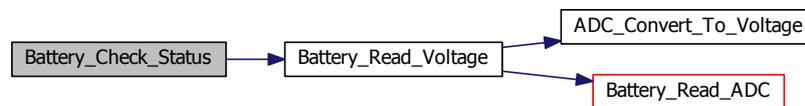
True (1) or False (0)

Definition at line 40 of file [Battery.c](#).

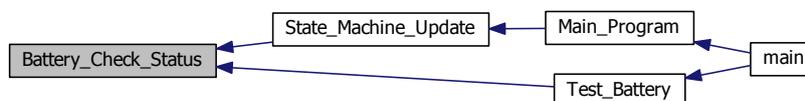
References [Battery_Read_Voltage\(\)](#), and [Battery_Safe_Voltage](#).

Referenced by [State_Machine_Update\(\)](#), and [Test_Battery\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.66.3.2 int Battery_Read_ADC () [inline]

This function is used to read the ADC port for the battery.

Returns

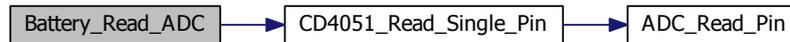
ADC value of the battery.

Definition at line 20 of file [Battery.c](#).

References [Battery_Mux_Port](#), and [CD4051_Read_Single_Pin\(\)](#).

Referenced by [Battery_Read_Voltage\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.66.3.3 float Battery_Read_Voltage (void)

This function is used to read the battery voltage.

Returns

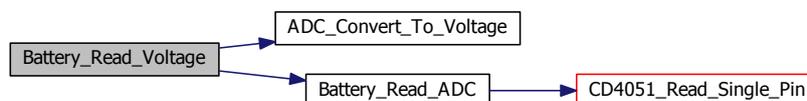
Voltage of the battery.

Definition at line 30 of file [Battery.c](#).

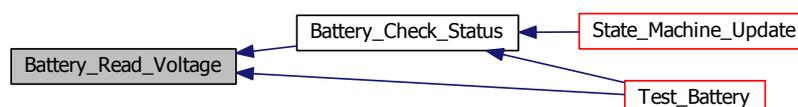
References [ADC_Convert_To_Voltage\(\)](#), and [Battery_Read_ADC\(\)](#).

Referenced by [Battery_Check_Status\(\)](#), and [Test_Battery\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.67 Battery.c

```

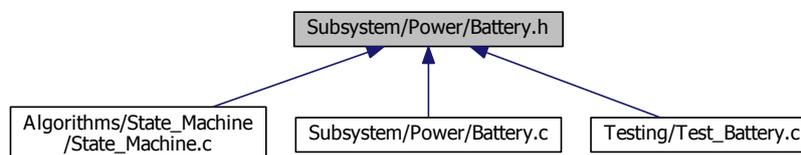
00001 /**
00002  * @file Battery.c
00003  * @author Nicholas Sikkema
00004  * @version Revision: 2.0
00005  * @date Last Updated: 1/22/2015\n Created: 11/13/2014 2:33:44 PM
00006  * @brief This file is code to read the battery voltage.
00007  */
00008
00009 /** @brief This is the lowest safe voltage value. */
00010 #define Battery_Safe_Voltage 3.45
00011
00012 #include "../Microcontroller/ADC/ADC.h"
00013 #include "../Microcontroller/ADC/CD4051.h"
00014 #include "Battery.h"
00015
00016 /**
00017  * @brief This function is used to read the ADC port for the battery.
00018  * @return ADC value of the battery.
00019  */
00020 inline int Battery_Read_ADC()
00021 {
00022     /* Return the ADC value for the battery. */
00023     return(CD4051_Read_Single_Pin(Battery_Mux_Port));
00024 }
00025
00026 /**
00027  * @brief This function is used to read the battery voltage.
00028  * @return Voltage of the battery.
00029  */
00030 float Battery_Read_Voltage()
00031 {
00032     /* Return the voltage value for the battery. */
00033     return(ADC_Convert_To_Voltage(Battery_Read_ADC()));
00034 }
00035
00036 /**
00037  * @brief This function is used to check to see if the battery is below X volts.
00038  * @return True (1) or False (0)
00039  */
00040 int Battery_Check_Status()
00041 {
00042     /* Return if the battery voltage is less than the safe voltage. */
00043     return(Battery_Read_Voltage() < Battery_Safe_Voltage);
00044 }

```

3.68 Subsystem/Power/Battery.h File Reference

This is the header file for the battery voltage reading code.

This graph shows which files directly or indirectly include this file:



Functions

- float [Battery_Read_Voltage](#) (void)
This function is used to read the battery voltage.
- int [Battery_Check_Status](#) (void)
This function is used to check to see if the battery is below X volts.

3.68.1 Detailed Description

This is the header file for the battery voltage reading code.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 1/22/2015

Created: 11/13/2014 2:34:28 PM

Definition in file [Battery.h](#).

3.68.2 Function Documentation

3.68.2.1 `int Battery_Check_Status (void)`

This function is used to check to see if the battery is below X volts.

Returns

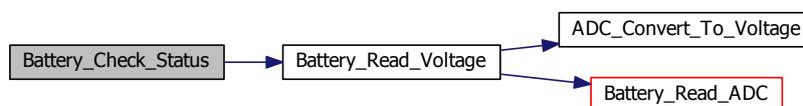
True (1) or False (0)

Definition at line 40 of file [Battery.c](#).

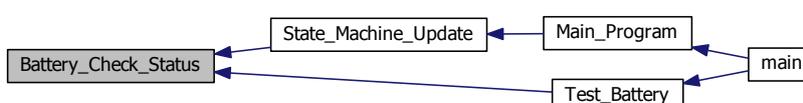
References [Battery_Read_Voltage\(\)](#), and [Battery_Safe_Voltage](#).

Referenced by [State_Machine_Update\(\)](#), and [Test_Battery\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.68.2.2 float Battery_Read_Voltage (void)

This function is used to read the battery voltage.

Returns

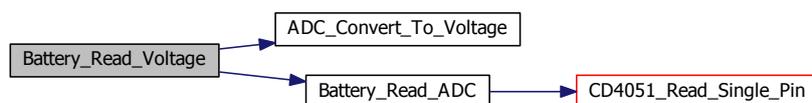
Voltage of the battery.

Definition at line 30 of file [Battery.c](#).

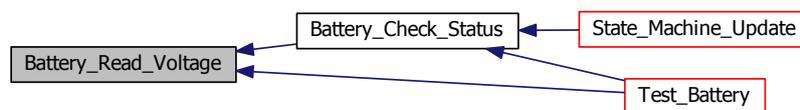
References [ADC_Convert_To_Voltage\(\)](#), and [Battery_Read_ADC\(\)](#).

Referenced by [Battery_Check_Status\(\)](#), and [Test_Battery\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.69 Battery.h

```

00001 /**
00002  * @file Battery.h
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 1/22/2015\n Created: 11/13/2014 2:34:28 PM
00006  * @brief This is the header file for the battery voltage reading code.
00007  */
00008
00009 #ifndef BATTERY_H_
00010 #define BATTERY_H_
00011
00012 float Battery_Read_Voltage(void);
00013 int Battery_Check_Status(void);
00014
00015 #endif /* BATTERY_H_ */
  
```

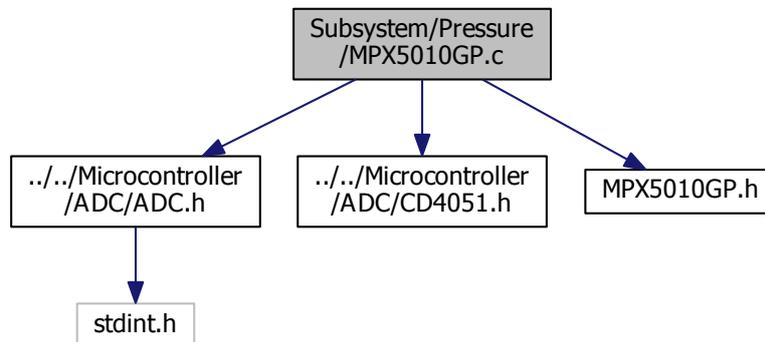
3.70 Subsystem/Pressure/MPX5010GP.c File Reference

This file is code to read the depth from the MPX5010GP pressure sensor.

```

#include "../Microcontroller/ADC/ADC.h"
#include "../Microcontroller/ADC/CD4051.h"
#include "MPX5010GP.h"
  
```

Include dependency graph for MPX5010GP.c:



Macros

- `#define VFSS 4.7`
The full scale span for the conversion between Voltage and PSI.
- `#define V_Offset 0.12`
The offset voltage for the conversion between Voltage and PSI.
- `#define Max_PSI 1.45`
Max rated PSI for the pressure sensor.

Functions

- `float MPX5010GP_Calculate_PSI (int ADC_Value)`
Converts the pressure sensor's ADC value to a depth in PSI.
- `int MPX5010GP_Read_Value ()`
Read the ADC with the correct multiplexer port selected.
- `float MPX5010GP_Depth ()`
Obtains the pressure sensor's depth in feet.

3.70.1 Detailed Description

This file is code to read the depth from the MPX5010GP pressure sensor.

Author

Nicholas Sikkema

Version

Revision: 3.0

Date

Last Updated: 2/11/2015
Created: 2/11/2014 11:50 PM

Definition in file [MPX5010GP.c](#).

3.70.2 Macro Definition Documentation

3.70.2.1 #define Max_PSI 1.45

Max rated PSI for the pressure sensor.

Definition at line 14 of file [MPX5010GP.c](#).

Referenced by [MPX5010GP_Calculate_PSI\(\)](#).

3.70.2.2 #define V_Offset 0.12

The offset voltage for the conversion between Voltage and PSI.

Definition at line 12 of file [MPX5010GP.c](#).

Referenced by [MPX5010GP_Calculate_PSI\(\)](#).

3.70.2.3 #define VFSS 4.7

The full scale span for the conversion between Voltage and PSI.

Definition at line 10 of file [MPX5010GP.c](#).

Referenced by [MPX5010GP_Calculate_PSI\(\)](#).

3.70.3 Function Documentation

3.70.3.1 float MPX5010GP_Calculate_PSI (int *ADC_Value*)

Converts the pressure sensor's ADC value to a depth in PSI.

Parameters

<i>ADC_Value</i>	The current ADC value for the pressure sensor.
------------------	--

Returns

The current pressure sensor depth in PSI.

Definition at line 25 of file [MPX5010GP.c](#).

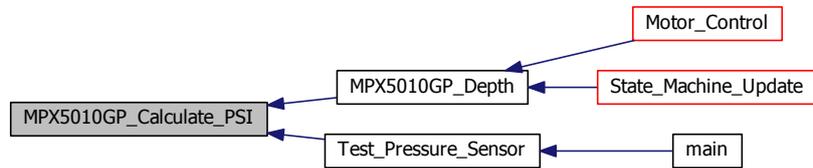
References [ADC_Convert_To_Voltage\(\)](#), [Max_PSI](#), [V_Offset](#), and [VFSS](#).

Referenced by [MPX5010GP_Depth\(\)](#), and [Test_Pressure_Sensor\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.70.3.2 float MPX5010GP_Depth (void)

Obtains the pressure sensor's depth in feet.

Returns

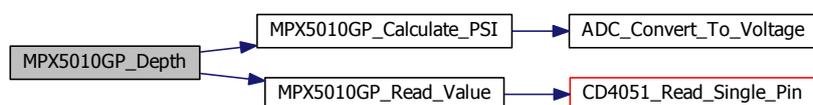
The current pressure sensor depth in feet.

Definition at line 45 of file [MPX5010GP.c](#).

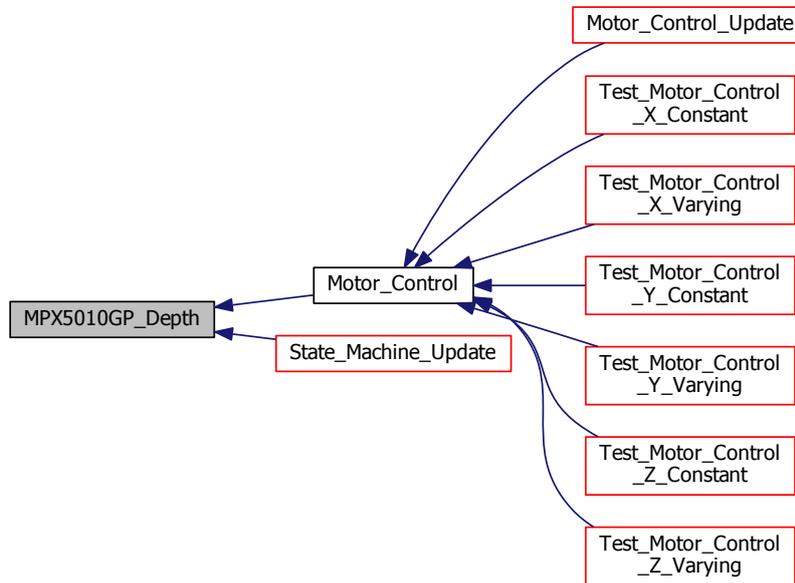
References [Foot_Water_Per_PSI](#), [MPX5010GP_Calculate_PSI\(\)](#), and [MPX5010GP_Read_Value\(\)](#).

Referenced by [Motor_Control\(\)](#), and [State_Machine_Update\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.70.3.3 int MPX5010GP_Read_Value () [inline]

Read the ADC with the correct multiplexer port selected.

Returns

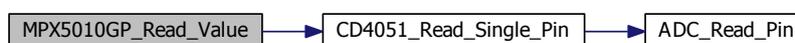
The current pressure sensor ADC value.

Definition at line 35 of file [MPX5010GP.c](#).

References [CD4051_Read_Single_Pin\(\)](#), and [Pressure_Sensor_Mux_Port](#).

Referenced by [MPX5010GP_Depth\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.71 MPX5010GP.c

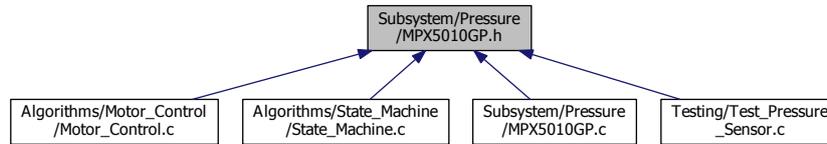
```

00001 /**
00002  * @file MPX5010GP.c
00003  * @author Nicholas Sikkema
00004  * @version Revision: 3.0
00005  * @date Last Updated: 2/11/2015\n Created: 2/11/2014 11:50 PM
00006  * @brief This file is code to read the depth from the MPX5010GP pressure sensor.
00007  */
00008
00009 /** @brief The full scale span for the conversion between Voltage and PSI. */
00010 #define VFSS 4.7
00011 /** @brief The offset voltage for the conversion between Voltage and PSI. */
00012 #define V_Offset 0.12
00013 /** @brief Max rated PSI for the pressure sensor. */
00014 #define Max_PSI 1.45
00015
00016 #include "../Microcontroller/ADC/ADC.h"
00017 #include "../Microcontroller/ADC/CD4051.h"
00018 #include "MPX5010GP.h"
00019
00020 /**
00021  * @brief Converts the pressure sensor's ADC value to a depth in PSI.
00022  * @param ADC_Value The current ADC value for the pressure sensor.
00023  * @return The current pressure sensor depth in PSI.
00024  */
00025 float MPX5010GP_Calculate_PSI(int ADC_Value)
00026 {
00027     /* Convert the ADC value to a PSI value. */
00028     return ((ADC_Convert_To_Voltage(ADC_Value)-V_Offset) *
00029             Max_PSI/VFSS);
00029 }
00030
00031 /**
00032  * @brief Read the ADC with the correct multiplexer port selected
00033  * @return The current pressure sensor ADC value.
00034  */
00035 inline int MPX5010GP_Read_Value()
00036 {
00037     /* Retrieve the ADC value for the pressure sensor. */
00038     return(CD4051_Read_Single_Pin(Pressure_Sensor_Mux_Port));
00039 }
00040
00041 /**
00042  * @brief Obtains the pressure sensor's depth in feet.
00043  * @return The current pressure sensor depth in feet.
00044  */
00045 float MPX5010GP_Depth()
00046 {
00047     /* Gather the PSI value and convert the value to a depth value. */
00048     return(MPX5010GP_Calculate_PSI(MPX5010GP_Read_Value()) *
00049             Foot_Water_Per_PSI);
00049 }
  
```

3.72 Subsystem/Pressure/MPX5010GP.h File Reference

This file is code to read the depth from the MPX5010GP pressure sensor.

This graph shows which files directly or indirectly include this file:



Macros

- `#define Foot_Water_Per_PSI 2.921`
The conversion factor between PSI and feet.

Functions

- float `MPX5010GP_Depth` (void)
Obtains the pressure sensor's depth in feet.
- float `MPX5010GP_Calculate_PSI` (int ADC_Value)
Converts the pressure sensor's ADC value to a depth in PSI.

3.72.1 Detailed Description

This file is code to read the depth from the MPX5010GP pressure sensor.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 2/11/2015
Created: 2/11/2014 11:50 PM

Definition in file [MPX5010GP.h](#).

3.72.2 Macro Definition Documentation

3.72.2.1 `#define Foot_Water_Per_PSI 2.921`

The conversion factor between PSI and feet.

Definition at line 14 of file [MPX5010GP.h](#).

Referenced by [MPX5010GP_Depth\(\)](#), and [Test_Pressure_Sensor\(\)](#).

3.72.3 Function Documentation

3.72.3.1 float MPX5010GP_Calculate_PSI (int *ADC_Value*)

Converts the pressure sensor's ADC value to a depth in PSI.

Parameters

<i>ADC_Value</i>	The current ADC value for the pressure sensor.
------------------	--

Returns

The current pressure sensor depth in PSI.

Definition at line 25 of file [MPX5010GP.c](#).

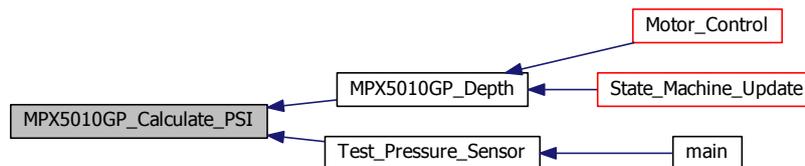
References [ADC_Convert_To_Voltage\(\)](#), [Max_PSI](#), [V_Offset](#), and [VFSS](#).

Referenced by [MPX5010GP_Depth\(\)](#), and [Test_Pressure_Sensor\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.72.3.2 float MPX5010GP_Depth (void)

Obtains the pressure sensor's depth in feet.

Returns

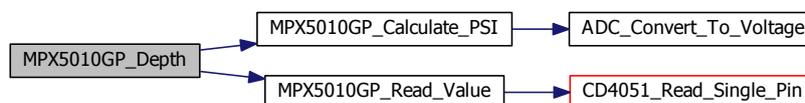
The current pressure sensor depth in feet.

Definition at line 45 of file [MPX5010GP.c](#).

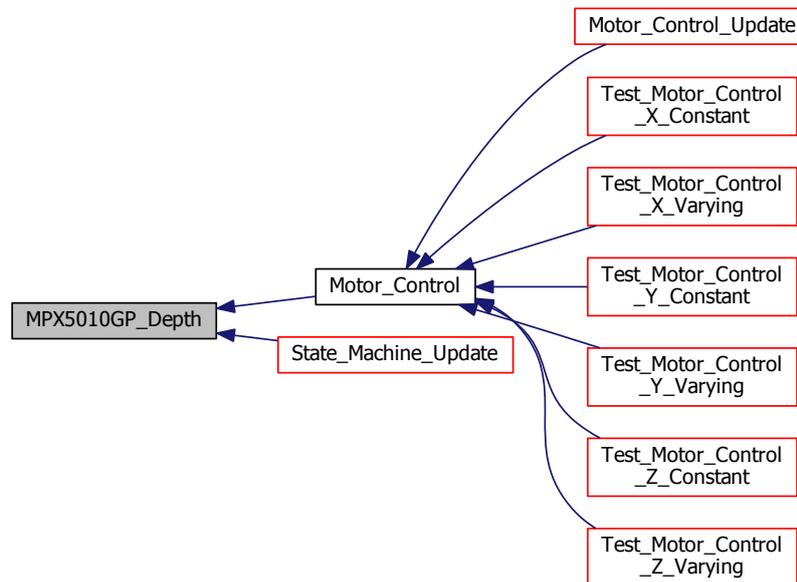
References [Foot_Water_Per_PSI](#), [MPX5010GP_Calculate_PSI\(\)](#), and [MPX5010GP_Read_Value\(\)](#).

Referenced by [Motor_Control\(\)](#), and [State_Machine_Update\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.73 MPX5010GP.h

```

00001 /**
00002  * @file MPX5010GP.h
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 2/11/2015\n Created: 2/11/2014 11:50 PM
00006  * @brief This file is code to read the depth from the MPX5010GP pressure sensor.
00007  */
00008
00009 #ifndef MPX5010GP_H_
00010 #define MPX5010GP_H_
00011
00012 #ifndef Foot_Water_Per_PSI
00013     /** @brief The conversion factor between PSI and feet. */
00014     #define Foot_Water_Per_PSI 2.921
00015 #endif
00016
00017 float MPX5010GP_Depth(void);
00018 float MPX5010GP_Calculate_PSI(int ADC_Value);
00019
00020 #endif /* MPX5010GP_H_ */
  
```

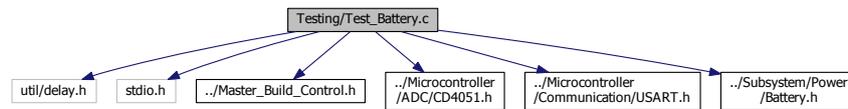
3.74 Testing/Test_Battery.c File Reference

This file is for the testing of the battery code.

```

#include <util/delay.h>
#include <stdio.h>
#include "../Master_Build_Control.h"
#include "../Microcontroller/ADC/CD4051.h"
#include "../Microcontroller/Communication/USART.h"
#include "../Subsystem/Power/Battery.h"
  
```

Include dependency graph for Test_Battery.c:



Macros

- `#define F_CPU 16000000UL`
The current clock speed for the microcontroller.

Functions

- `void Test_Battery (void)`
This function is used to test the reading of the battery code.

3.74.1 Detailed Description

This file is for the testing of the battery code.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 4/29/2015
 Created: 2/28/2015 6:38:56 PM

Definition in file [Test_Battery.c](#).

3.74.2 Macro Definition Documentation

3.74.2.1 `#define F_CPU 16000000UL`

The current clock speed for the microcontroller.

Definition at line 11 of file [Test_Battery.c](#).

3.74.3 Function Documentation

3.74.3.1 `void Test_Battery (void)`

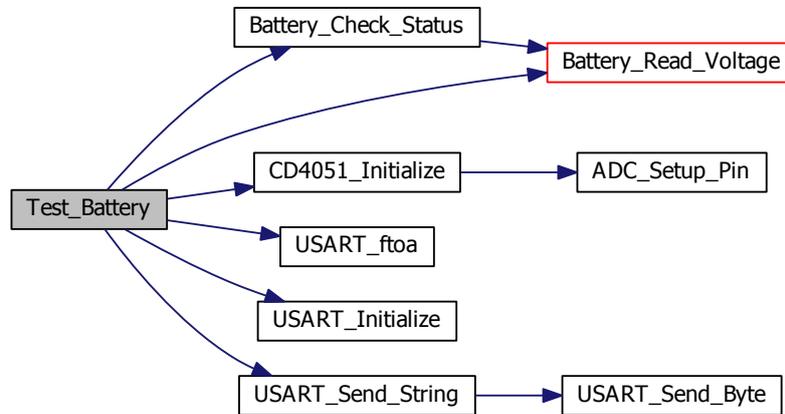
This function is used to test the reading of the battery code.

Definition at line 24 of file [Test_Battery.c](#).

References [Battery_Check_Status\(\)](#), [Battery_Read_Voltage\(\)](#), [CD4051_Initialize\(\)](#), [FLOATING_POINT_BUFFER_SIZE](#), [USART_ftoa\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.75 Test_Battery.c

```

00001 /**
00002  * @file Test_Battery.c
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 4/29/2015\n Created: 2/28/2015 6:38:56 PM
00006  * @brief This file is for the testing of the battery code.
00007  */
00008
00009 #ifndef F_CPU
00010     /** @brief The current clock speed for the microcontroller. */
00011     #define F_CPU 16000000UL
00012 #endif
00013
00014 #include <util/delay.h>
00015 #include <stdio.h>
00016 #include "../Master_Build_Control.h"
00017 #include "../Microcontroller/ADC/CD4051.h"
00018 #include "../Microcontroller/Communication/USART.h"
00019 #include "../Subsystem/Power/Battery.h"
00020
00021 /**
00022  * @brief This function is used to test the reading of the battery code.
00023  */
00024 void Test_Battery(void)
  
```

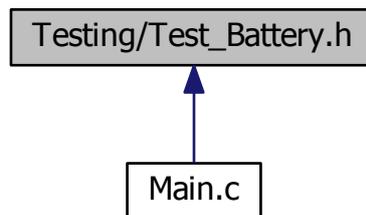
```

00025 {
00026     /* Initialize the multiplexer. */
00027     CD4051_Initialize();
00028     /* Initialize USART communication. */
00029     USART_Initialize();
00030     /* Initialize the temporary variables. */
00031     char Buffer_ADC_Value_Voltage[FLOATING_POINT_BUFFER_SIZE] = {0x00};
00032     char Buffer[80] = {0x00};
00033     while(1)
00034     {
00035         float ADC_Value_Voltage = Battery_Read_Voltage();
00036         if(Battery_Check_Status() == 0)
00037         {
00038             USART_Send_String("Status: Good\r\n");
00039         }
00040         else
00041         {
00042             USART_Send_String("Status: Bad\r\n");
00043         }
00044         sprintf(Buffer,"%s V\r\n", USART_ftoa(ADC_Value_Voltage, Buffer_ADC_Value_Voltage));
00045         USART_Send_String(Buffer);
00046         _delay_ms(500);
00047     }
00048 }

```

3.76 Testing/Test_Battery.h File Reference

This graph shows which files directly or indirectly include this file:



Functions

- void [Test_Battery](#) (void)

This function is used to test the reading of the battery code.

3.76.1 Function Documentation

3.76.1.1 void Test_Battery (void)

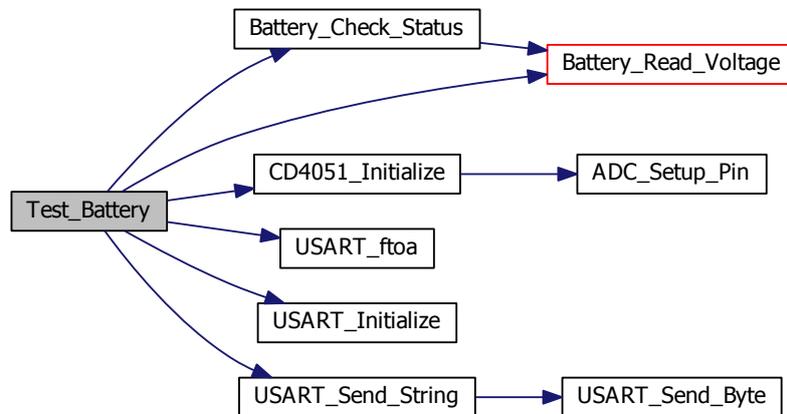
This function is used to test the reading of the battery code.

Definition at line 24 of file [Test_Battery.c](#).

References [Battery_Check_Status\(\)](#), [Battery_Read_Voltage\(\)](#), [CD4051_Initialize\(\)](#), [FLOATING_POINT_BUFFER_SIZE](#), [USART_ftoa\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.77 Test_Battery.h

```

00001 /**
00002  * @file Test_Multiplexer.h
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 4/29/2015\n Created: 2/28/2015 6:39:10 PM
00006  * @brief This file is the header file for the testing of the battery code.
00007  */
00008
00009 #ifndef TEST_BATTERY_H_
00010 #define TEST_BATTERY_H_
00011
00012 void Test_Battery(void);
00013
00014 #endif /* TEST_BATTERY_H_ */
  
```

3.78 Testing/Test_H_Bridge.c File Reference

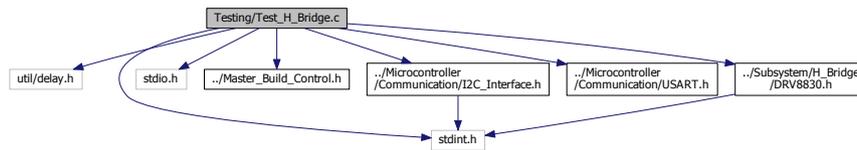
This file is for the testing of the I2C H-Bridge code.

```

#include <util/delay.h>
#include <stdint.h>
#include <stdio.h>
#include "../Master_Build_Control.h"
#include "../Microcontroller/Communication/I2C_Interface.h"
#include "../Microcontroller/Communication/USART.h"
#include "../Subsystem/H_Bridge/DRV8830.h"

```

Include dependency graph for Test_H_Bridge.c:



Macros

- `#define F_CPU 16000000UL`
The current clock speed for the microcontroller.

Functions

- void `Test_H_Bridge_Motor_X_Increase` (void)
- void `Test_H_Bridge_Motor_X_Decrease` (void)
- void `Test_H_Bridge_Motor_Y_Increase` (void)
- void `Test_H_Bridge_Motor_Y_Decrease` (void)
- void `Test_H_Bridge_Motor_Z_Increase` (void)
- void `Test_H_Bridge_Motor_Z_Decrease` (void)
- void `Test_H_Bridge_Motor_ALL_Increase` (void)
- void `Test_H_Bridge_Motor_ALL_Decrease` (void)
- void `Test_H_Bridge` (void)

This function is used to test the h-bridge code.

3.78.1 Detailed Description

This file is for the testing of the I2C H-Bridge code.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 4/29/2015
Created: 2/28/2015 6:35:33 PM

Definition in file [Test_H_Bridge.c](#).

3.78.2 Macro Definition Documentation

3.78.2.1 #define F_CPU 16000000UL

The current clock speed for the microcontroller.

Definition at line 11 of file [Test_H_Bridge.c](#).

3.78.3 Function Documentation

3.78.3.1 void Test_H_Bridge (void)

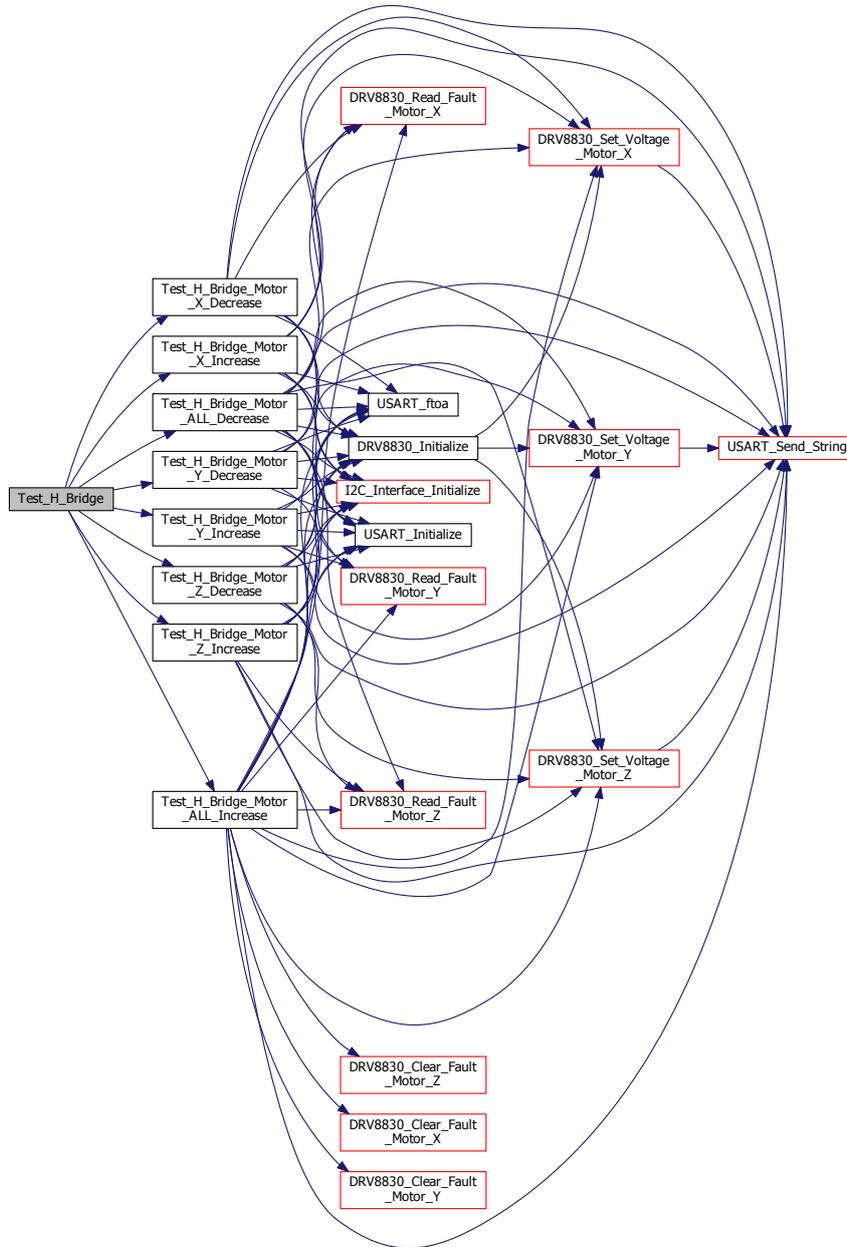
This function is used to test the h-bridge code.

Definition at line 212 of file [Test_H_Bridge.c](#).

References [Test_H_Bridge_Motor_ALL_Decrease\(\)](#), [Test_H_Bridge_Motor_ALL_Increase\(\)](#), [Test_H_Bridge_Motor_X_Decrease\(\)](#), [Test_H_Bridge_Motor_X_Increase\(\)](#), [Test_H_Bridge_Motor_Y_Decrease\(\)](#), [Test_H_Bridge_Motor_Y_Increase\(\)](#), [Test_H_Bridge_Motor_Z_Decrease\(\)](#), and [Test_H_Bridge_Motor_Z_Increase\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



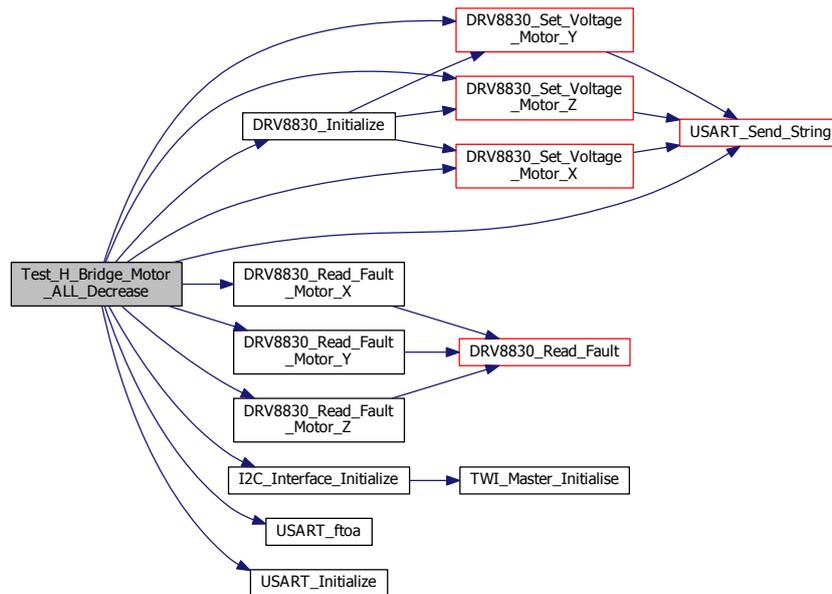
3.78.3.2 void Test_H_Bridge_Motor_ALL_Decrease (void)

Definition at line 180 of file Test_H_Bridge.c.

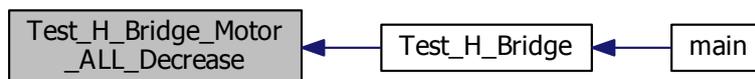
References [DRV8830_Initialize\(\)](#), [DRV8830_Read_Fault_Motor_X\(\)](#), [DRV8830_Read_Fault_Motor_Y\(\)](#), [DRV8830_Read_Fault_Motor_Z\(\)](#), [DRV8830_Set_Voltage_Motor_X\(\)](#), [DRV8830_Set_Voltage_Motor_Y\(\)](#), [DRV8830_Set_Voltage_Motor_Z\(\)](#), [FLOATING_POINT_BUFFER_SIZE](#), [I2C_Interface_Initialize\(\)](#), [Motor_Max_Voltage](#), [USART_ftoa\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_H_Bridge\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



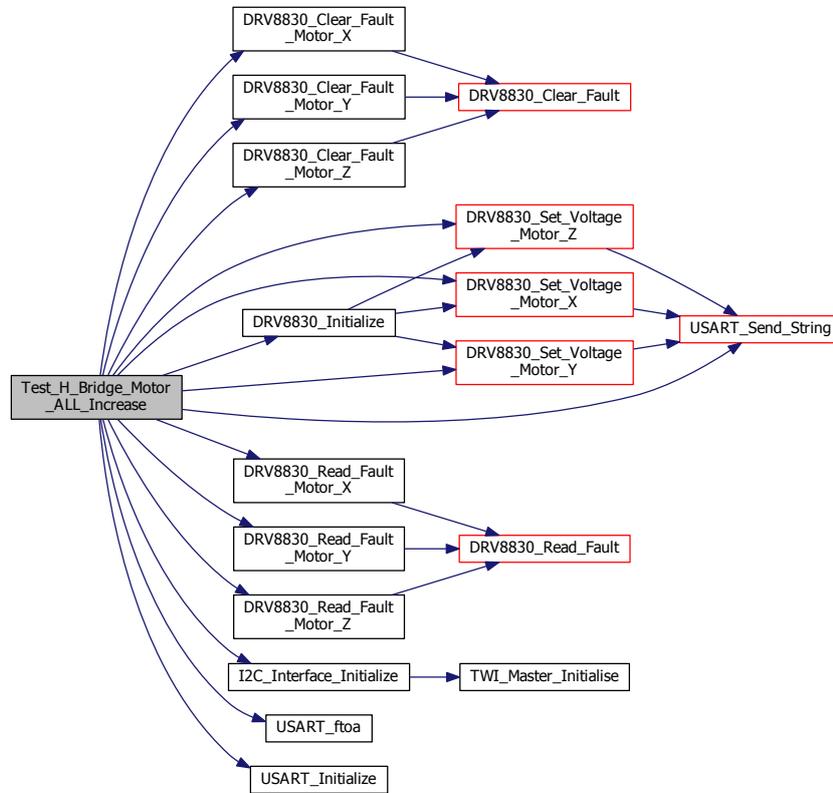
3.78.3.3 void Test_H_Bridge_Motor_ALL_Increase (void)

Definition at line 148 of file Test_H_Bridge.c.

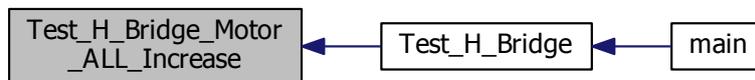
References [DRV8830_Clear_Fault_Motor_X\(\)](#), [DRV8830_Clear_Fault_Motor_Y\(\)](#), [DRV8830_Clear_Fault_Motor_Z\(\)](#), [DRV8830_Initialize\(\)](#), [DRV8830_Read_Fault_Motor_X\(\)](#), [DRV8830_Read_Fault_Motor_Y\(\)](#), [DRV8830_Read_Fault_Motor_Z\(\)](#), [DRV8830_Set_Voltage_Motor_X\(\)](#), [DRV8830_Set_Voltage_Motor_Y\(\)](#), [DRV8830_Set_Voltage_Motor_Z\(\)](#), [FLOATING_POINT_BUFFER_SIZE](#), [I2C_Interface_Initialize\(\)](#), [Motor_Max_Voltage](#), [USART_ftoa\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_H_Bridge\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



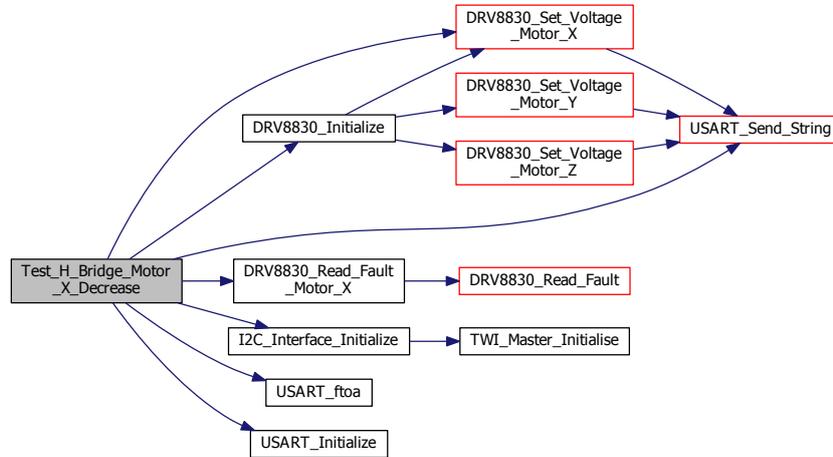
3.78.3.4 void Test_H_Bridge_Motor_X_Decrease (void)

Definition at line 43 of file [Test_H_Bridge.c](#).

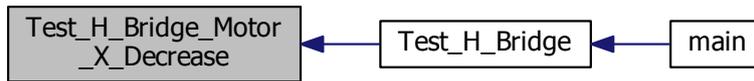
References [DRV8830_Initialize\(\)](#), [DRV8830_Read_Fault_Motor_X\(\)](#), [DRV8830_Set_Voltage_Motor_X\(\)](#), [FLOATING_POINT_BUFFER_SIZE](#), [I2C_Interface_Initialize\(\)](#), [Motor_Max_Voltage](#), [USART_ftoa\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_H_Bridge\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



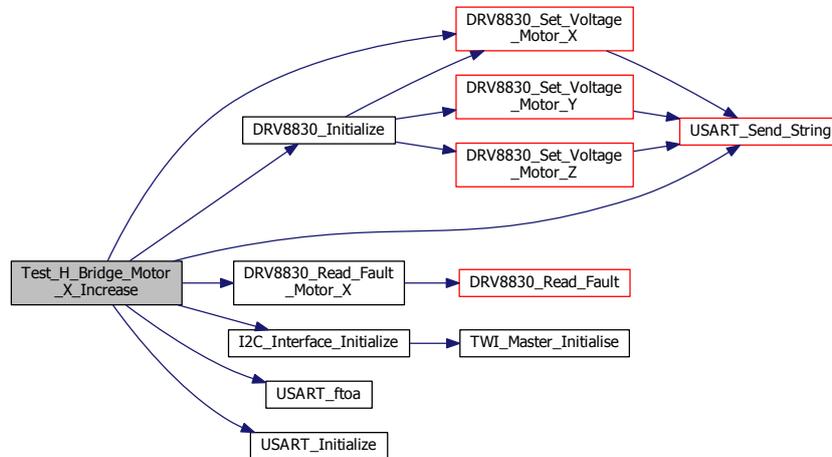
3.78.3.5 void Test_H_Bridge_Motor_X_Increase (void)

Definition at line 22 of file [Test_H_Bridge.c](#).

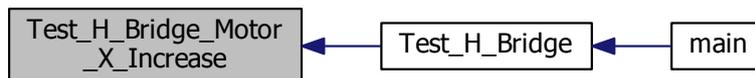
References [DRV8830_Initialize\(\)](#), [DRV8830_Read_Fault_Motor_X\(\)](#), [DRV8830_Set_Voltage_Motor_X\(\)](#), [FLOATING_POINT_BUFFER_SIZE](#), [I2C_Interface_Initialize\(\)](#), [Motor_Max_Voltage](#), [USART_ftoa\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_H_Bridge\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



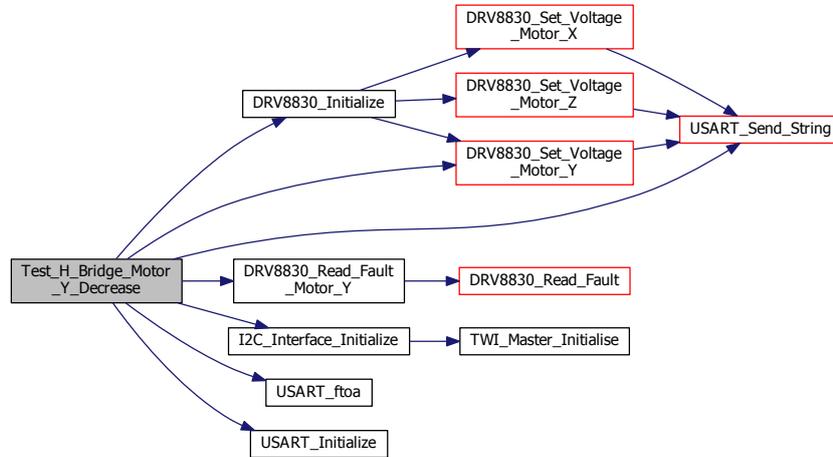
3.78.3.6 void Test_H_Bridge_Motor_Y_Decrease (void)

Definition at line 85 of file [Test_H_Bridge.c](#).

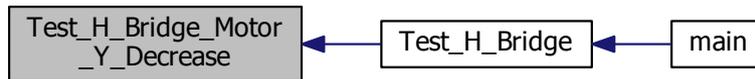
References [DRV8830_Initialize\(\)](#), [DRV8830_Read_Fault_Motor_Y\(\)](#), [DRV8830_Set_Voltage_Motor_Y\(\)](#), [FLOATING_POINT_BUFFER_SIZE](#), [I2C_Interface_Initialize\(\)](#), [Motor_Max_Voltage](#), [USART_ftoa\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_H_Bridge\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



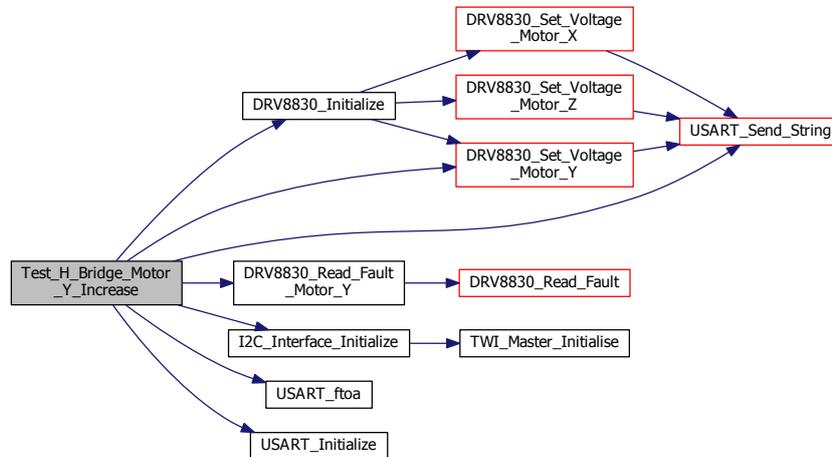
3.78.3.7 void Test_H_Bridge_Motor_Y_Increase (void)

Definition at line 64 of file [Test_H_Bridge.c](#).

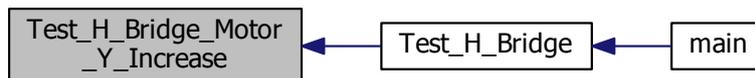
References [DRV8830_Initialize\(\)](#), [DRV8830_Read_Fault_Motor_Y\(\)](#), [DRV8830_Set_Voltage_Motor_Y\(\)](#), [FLOATING_POINT_BUFFER_SIZE](#), [I2C_Interface_Initialize\(\)](#), [Motor_Max_Voltage](#), [USART_ftoa\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_H_Bridge\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



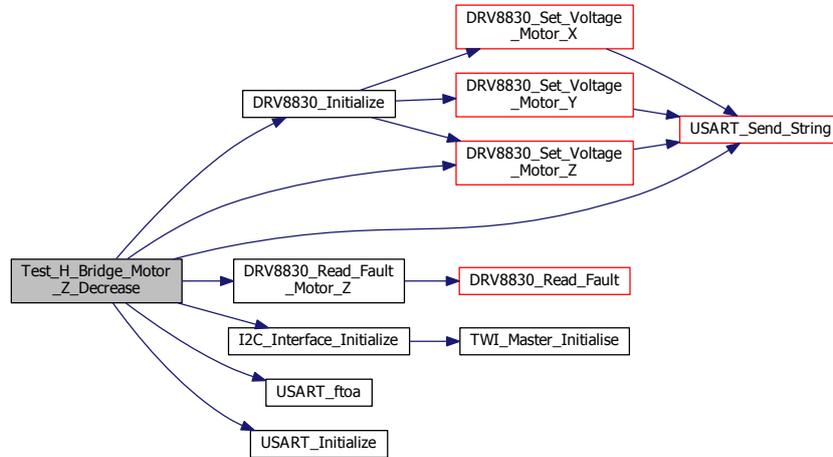
3.78.3.8 void Test_H_Bridge_Motor_Z_Decrease (void)

Definition at line 127 of file [Test_H_Bridge.c](#).

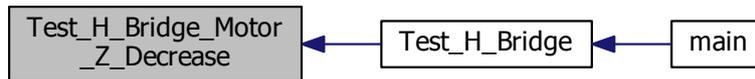
References [DRV8830_Initialize\(\)](#), [DRV8830_Read_Fault_Motor_Z\(\)](#), [DRV8830_Set_Voltage_Motor_Z\(\)](#), [FLOATING_POINT_BUFFER_SIZE](#), [I2C_Interface_Initialize\(\)](#), [Motor_Max_Voltage](#), [USART_ftoa\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_H_Bridge\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



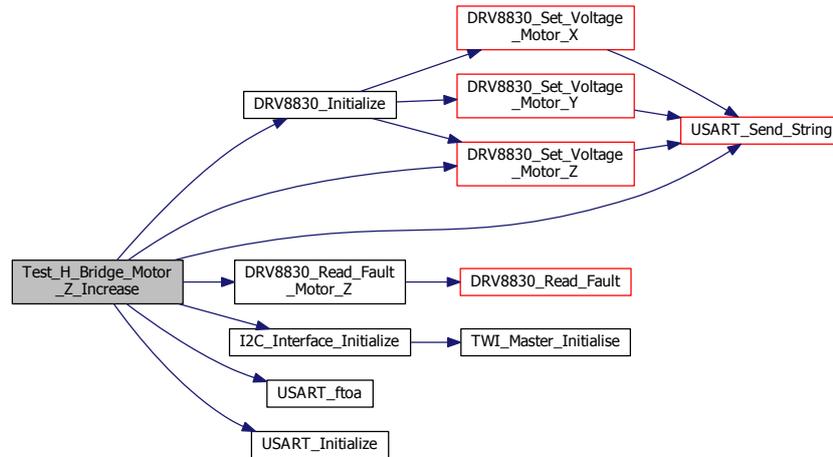
3.78.3.9 void Test_H_Bridge_Motor_Z_Increase (void)

Definition at line 106 of file [Test_H_Bridge.c](#).

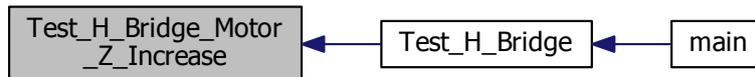
References [DRV8830_Initialize\(\)](#), [DRV8830_Read_Fault_Motor_Z\(\)](#), [DRV8830_Set_Voltage_Motor_Z\(\)](#), [FLOATING_POINT_BUFFER_SIZE](#), [I2C_Interface_Initialize\(\)](#), [Motor_Max_Voltage](#), [USART_ftoa\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_H_Bridge\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.79 Test_H_Bridge.c

```

00001 /**
00002  * @file Test_H_Bridge.c
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 4/29/2015 \n Created: 2/28/2015 6:35:33 PM
00006  * @brief This file is for the testing of the I2C H-Bridge code.
00007  */
00008
00009 #ifndef F_CPU
00010     /** @brief The current clock speed for the microcontroller. */
00011     #define F_CPU 16000000UL
00012 #endif
00013
00014 #include <util/delay.h>
00015 #include <stdint.h>
00016 #include <stdio.h>
00017 #include "../Master_Build_Control.h"
00018 #include "../Microcontroller/Communication/I2C_Interface.h"
00019 #include "../Microcontroller/Communication/USART.h"
00020 #include "../Subsystem/H_Bridge/DRV8830.h"
00021
00022 void Test_H_Bridge_Motor_X_Increase(void)
00023 {
00024     I2C_Interface_Initialize();
00025     USART_Initialize();
00026     DRV8830_Initialize();
00027     uint8_t fault = 0;
00028     char Buffer_i[FLOATING_POINT_BUFFER_SIZE] = {0x00};
00029     char Buffer[50] = {0x00};
00030     while(1)
00031     {
  
```

```

00032     for(float i=-Motor_Max_Voltage; i<Motor_Max_Voltage; i+=0.07)
00033     {
00034         DRV8830_Set_Voltage_Motor_X(i);
00035         fault = DRV8830_Read_Fault_Motor_X();
00036         sprintf(Buffer,"Voltage: %s Fault: %d\r\n", USART_ftoa(i, Buffer_i), fault);
00037         USART_Send_String(Buffer);
00038         _delay_ms(500);
00039     }
00040 }
00041 }
00042
00043 void Test_H_Bridge_Motor_X_Decrease(void)
00044 {
00045     I2C_Interface_Initialize();
00046     USART_Initialize();
00047     DRV8830_Initialize();
00048     uint8_t fault = 0;
00049     char Buffer_i[FLOATING_POINT_BUFFER_SIZE] = {0x00};
00050     char Buffer[50] = {0x00};
00051     while(1)
00052     {
00053         for(float i=Motor_Max_Voltage; i>-Motor_Max_Voltage; i-=0.07)
00054         {
00055             DRV8830_Set_Voltage_Motor_X(i);
00056             fault = DRV8830_Read_Fault_Motor_X();
00057             sprintf(Buffer,"Voltage: %s Fault: %d\r\n", USART_ftoa(i, Buffer_i), fault);
00058             USART_Send_String(Buffer);
00059             _delay_ms(500);
00060         }
00061     }
00062 }
00063
00064 void Test_H_Bridge_Motor_Y_Increase(void)
00065 {
00066     I2C_Interface_Initialize();
00067     USART_Initialize();
00068     DRV8830_Initialize();
00069     uint8_t fault = 0;
00070     char Buffer_i[FLOATING_POINT_BUFFER_SIZE] = {0x00};
00071     char Buffer[50] = {0x00};
00072     while(1)
00073     {
00074         for(float i=-Motor_Max_Voltage; i<Motor_Max_Voltage; i+=0.07)
00075         {
00076             DRV8830_Set_Voltage_Motor_Y(i);
00077             fault = DRV8830_Read_Fault_Motor_Y();
00078             sprintf(Buffer,"Voltage: %s Fault: %d\r\n", USART_ftoa(i, Buffer_i), fault);
00079             USART_Send_String(Buffer);
00080             _delay_ms(500);
00081         }
00082     }
00083 }
00084
00085 void Test_H_Bridge_Motor_Y_Decrease(void)
00086 {
00087     I2C_Interface_Initialize();
00088     USART_Initialize();
00089     DRV8830_Initialize();
00090     uint8_t fault = 0;
00091     char Buffer_i[FLOATING_POINT_BUFFER_SIZE] = {0x00};
00092     char Buffer[50] = {0x00};
00093     while(1)
00094     {
00095         for(float i=Motor_Max_Voltage; i>-Motor_Max_Voltage; i-=0.07)
00096         {
00097             DRV8830_Set_Voltage_Motor_Y(i);
00098             fault = DRV8830_Read_Fault_Motor_Y();
00099             sprintf(Buffer,"Voltage: %s Fault: %d\r\n", USART_ftoa(i, Buffer_i), fault);
00100             USART_Send_String(Buffer);
00101             _delay_ms(500);
00102         }
00103     }
00104 }
00105
00106 void Test_H_Bridge_Motor_Z_Increase(void)
00107 {
00108     I2C_Interface_Initialize();
00109     USART_Initialize();
00110     DRV8830_Initialize();
00111     uint8_t fault = 0;
00112     char Buffer_i[FLOATING_POINT_BUFFER_SIZE] = {0x00};
00113     char Buffer[50] = {0x00};
00114     while(1)
00115     {
00116         for(float i=-Motor_Max_Voltage; i<Motor_Max_Voltage; i+=0.07)
00117         {
00118             DRV8830_Set_Voltage_Motor_Z(i);

```

```

00119         fault = DRV8830_Read_Fault_Motor_Z();
00120         sprintf(Buffer, "Voltage: %s Fault: %d\r\n", USART_ftoa(i, Buffer_i), fault);
00121         USART_Send_String(Buffer);
00122         _delay_ms(1500);
00123     }
00124 }
00125 }
00126
00127 void Test_H_Bridge_Motor_Z_Decrease(void)
00128 {
00129     I2C_Interface_Initialize();
00130     USART_Initialize();
00131     DRV8830_Initialize();
00132     uint8_t fault = 0;
00133     char Buffer_i[FLOATING_POINT_BUFFER_SIZE] = {0x00};
00134     char Buffer[50] = {0x00};
00135     while(1)
00136     {
00137         for(float i=Motor_Max_Voltage; i>=-Motor_Max_Voltage; i-=0.07)
00138         {
00139             DRV8830_Set_Voltage_Motor_Z(i);
00140             fault = DRV8830_Read_Fault_Motor_Z();
00141             sprintf(Buffer, "Voltage: %s Fault: %d\r\n", USART_ftoa(i, Buffer_i), fault);
00142             USART_Send_String(Buffer);
00143             _delay_ms(500);
00144         }
00145     }
00146 }
00147
00148 void Test_H_Bridge_Motor_ALL_Increase(void)
00149 {
00150     I2C_Interface_Initialize();
00151     USART_Initialize();
00152     DRV8830_Initialize();
00153     uint8_t fault = 0;
00154     char Buffer_i[FLOATING_POINT_BUFFER_SIZE] = {0x00};
00155     char Buffer[50] = {0x00};
00156     while(1)
00157     {
00158         for(float i=-Motor_Max_Voltage; i<=Motor_Max_Voltage; i+=0.07)
00159         {
00160             DRV8830_Set_Voltage_Motor_X(i);
00161             fault = DRV8830_Read_Fault_Motor_X();
00162             DRV8830_Clear_Fault_Motor_X();
00163             sprintf(Buffer, "Voltage: %s Fault: %d\r\n", USART_ftoa(i, Buffer_i), fault);
00164             USART_Send_String(Buffer);
00165             DRV8830_Set_Voltage_Motor_Y(i);
00166             DRV8830_Clear_Fault_Motor_Y();
00167             fault = DRV8830_Read_Fault_Motor_Y();
00168             sprintf(Buffer, "Voltage: %s Fault: %d\r\n", USART_ftoa(i, Buffer_i), fault);
00169             USART_Send_String(Buffer);
00170             DRV8830_Set_Voltage_Motor_Z(i);
00171             DRV8830_Clear_Fault_Motor_Z();
00172             fault = DRV8830_Read_Fault_Motor_Z();
00173             sprintf(Buffer, "Voltage: %s Fault: %d\r\n", USART_ftoa(i, Buffer_i), fault);
00174             USART_Send_String(Buffer);
00175             _delay_ms(500);
00176         }
00177     }
00178 }
00179
00180 void Test_H_Bridge_Motor_ALL_Decrease(void)
00181 {
00182     I2C_Interface_Initialize();
00183     USART_Initialize();
00184     DRV8830_Initialize();
00185     uint8_t fault = 0;
00186     char Buffer_i[FLOATING_POINT_BUFFER_SIZE] = {0x00};
00187     char Buffer[50] = {0x00};
00188     while(1)
00189     {
00190         for(float i=Motor_Max_Voltage; i>=-Motor_Max_Voltage; i-=0.07)
00191         {
00192             DRV8830_Set_Voltage_Motor_X(i);
00193             fault = DRV8830_Read_Fault_Motor_X();
00194             sprintf(Buffer, "Voltage: %s Fault: %d\r\n", USART_ftoa(i, Buffer_i), fault);
00195             USART_Send_String(Buffer);
00196             DRV8830_Set_Voltage_Motor_Y(i);
00197             fault = DRV8830_Read_Fault_Motor_Y();
00198             sprintf(Buffer, "Voltage: %s Fault: %d\r\n", USART_ftoa(i, Buffer_i), fault);
00199             USART_Send_String(Buffer);
00200             DRV8830_Set_Voltage_Motor_Z(i);
00201             fault = DRV8830_Read_Fault_Motor_Z();
00202             sprintf(Buffer, "Voltage: %s Fault: %d\r\n", USART_ftoa(i, Buffer_i), fault);
00203             USART_Send_String(Buffer);
00204             _delay_ms(500);
00205         }

```

```

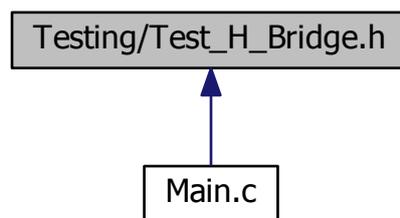
00206     }
00207 }
00208
00209 /**
00210  * @brief This function is used to test the h-bridge code.
00211  */
00212 void Test_H_Bridge(void)
00213 {
00214     #if H_Bridge_Test_Mode == 0
00215         #if H_Bridge_Test_Direction == 0
00216             Test_H_Bridge_Motor_X_Increase();
00217         #elif H_Bridge_Test_Direction == 1
00218             Test_H_Bridge_Motor_X_Decrease();
00219         #endif
00220     #elif H_Bridge_Test_Mode == 1
00221         #if H_Bridge_Test_Direction == 0
00222             Test_H_Bridge_Motor_Y_Increase();
00223         #elif H_Bridge_Test_Direction == 1
00224             Test_H_Bridge_Motor_Y_Decrease();
00225         #endif
00226     #elif H_Bridge_Test_Mode == 2
00227         #if H_Bridge_Test_Direction == 0
00228             Test_H_Bridge_Motor_Z_Increase();
00229         #elif H_Bridge_Test_Direction == 1
00230             Test_H_Bridge_Motor_Z_Decrease();
00231         #endif
00232     #elif H_Bridge_Test_Mode == 3
00233         #if H_Bridge_Test_Direction == 0
00234             Test_H_Bridge_Motor_ALL_Increase();
00235         #elif H_Bridge_Test_Direction == 1
00236             Test_H_Bridge_Motor_ALL_Decrease();
00237         #endif
00238     #endif
00239 }

```

3.80 Testing/Test_H_Bridge.h File Reference

This file is the header file for the testing of the H-Bridge code.

This graph shows which files directly or indirectly include this file:



Functions

- int [Test_H_Bridge](#) (void)
This function is used to test the h-bridge code.

3.80.1 Detailed Description

This file is the header file for the testing of the H-Bridge code.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 4/29/2015

Created: 2/28/2015 7:30:52 PM

Definition in file [Test_H_Bridge.h](#).

3.80.2 Function Documentation

3.80.2.1 `int Test_H_Bridge (void)`

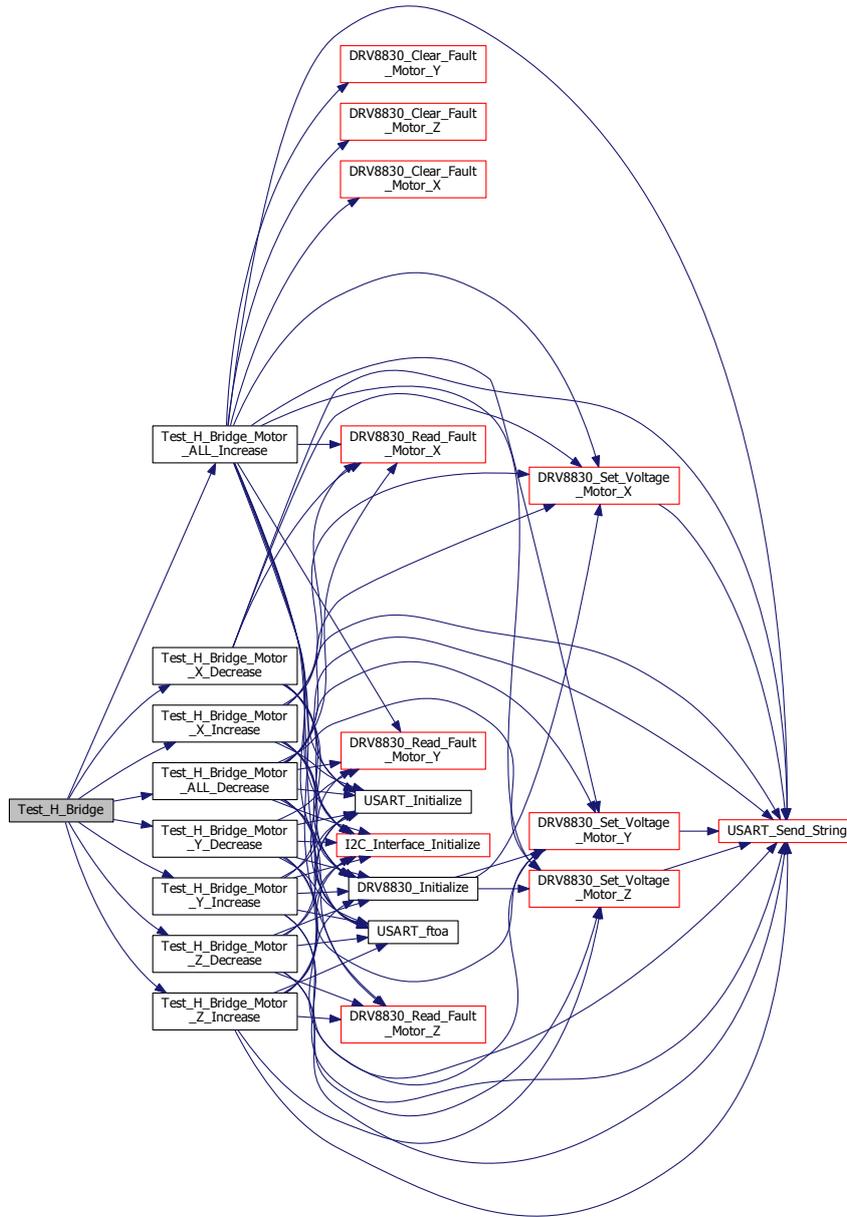
This function is used to test the h-bridge code.

Definition at line 212 of file [Test_H_Bridge.c](#).

References [Test_H_Bridge_Motor_ALL_Decrease\(\)](#), [Test_H_Bridge_Motor_ALL_Increase\(\)](#), [Test_H_Bridge_Motor_X_Decrease\(\)](#), [Test_H_Bridge_Motor_X_Increase\(\)](#), [Test_H_Bridge_Motor_Y_Decrease\(\)](#), [Test_H_Bridge_Motor_Y_Increase\(\)](#), [Test_H_Bridge_Motor_Z_Decrease\(\)](#), and [Test_H_Bridge_Motor_Z_Increase\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.81 Test_H_Bridge.h

```

00001 /**
00002  * @file Test_H_Bridge.h
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 4/29/2015\n Created: 2/28/2015 7:30:52 PM
00006  * @brief This file is the header file for the testing of the H-Bridge code.
00007  */
00008
00009 #ifndef TEST_H_BRIDGE_H_
00010 #define TEST_H_BRIDGE_H_
00011
00012 int Test_H_Bridge(void);
00013
00014 #endif /* TEST_H_BRIDGE_H_ */

```

3.82 Testing/Test_I2C_Compass.c File Reference

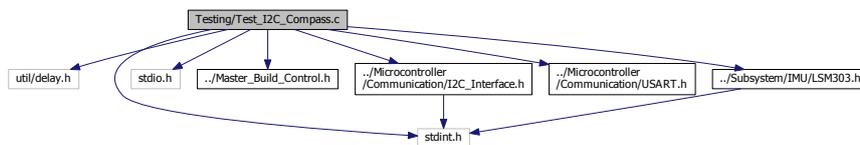
This file is for the testing of the I2C Compass code.

```

#include <util/delay.h>
#include <stdint.h>
#include <stdio.h>
#include "../Master_Build_Control.h"
#include "../Microcontroller/Communication/I2C_Interface.h"
#include "../Microcontroller/Communication/USART.h"
#include "../Subsystem/IMU/LSM303.h"

```

Include dependency graph for Test_I2C_Compass.c:



Macros

- #define `F_CPU` 16000000UL
The current clock speed for the microcontroller.

Functions

- void `Test_I2C_Compass_Mag_Acc` (void)
- void `Test_I2C_Compass_Heading` (void)
- void `Test_I2C_Compass` (void)
This function is used to test the compass code.

3.82.1 Detailed Description

This file is for the testing of the I2C Compass code.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 4/29/2015

Created: 2/28/2015 6:35:33 PM

Definition in file [Test_I2C_Compass.c](#).**3.82.2 Macro Definition Documentation****3.82.2.1 #define F_CPU 16000000UL**

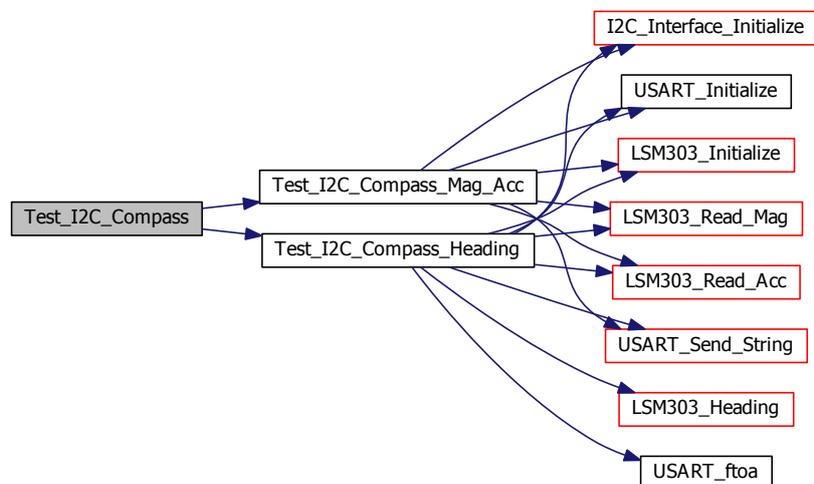
The current clock speed for the microcontroller.

Definition at line 11 of file [Test_I2C_Compass.c](#).**3.82.3 Function Documentation****3.82.3.1 void Test_I2C_Compass (void)**

This function is used to test the compass code.

Definition at line 64 of file [Test_I2C_Compass.c](#).References [Test_I2C_Compass_Heading\(\)](#), and [Test_I2C_Compass_Mag_Acc\(\)](#).Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



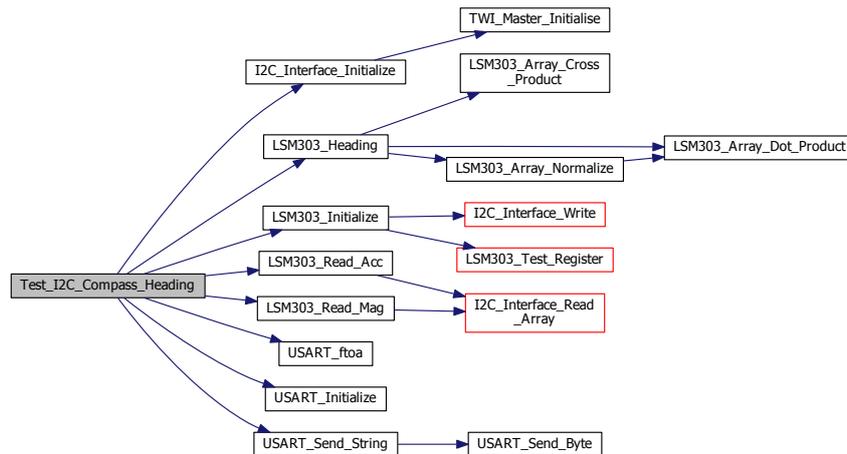
3.82.3.2 void Test_I2C_Compass_Heading (void)

Definition at line 40 of file [Test_I2C_Compass.c](#).

References [FLOATING_POINT_BUFFER_SIZE](#), [I2C_Interface_Initialize\(\)](#), [LSM303_Heading\(\)](#), [LSM303_Initialize\(\)](#), [LSM303_Read_Acc\(\)](#), [LSM303_Read_Mag\(\)](#), [USART_ftoa\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_I2C_Compass\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



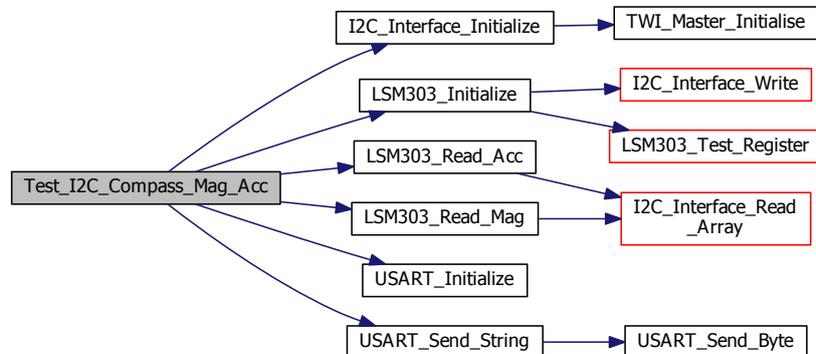
3.82.3.3 void Test_I2C_Compass_Mag_Acc (void)

Definition at line 22 of file [Test_I2C_Compass.c](#).

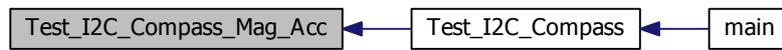
References [I2C_Interface_Initialize\(\)](#), [LSM303_Initialize\(\)](#), [LSM303_Read_Acc\(\)](#), [LSM303_Read_Mag\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_I2C_Compass\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.83 Test_I2C_Compass.c

```

00001 /**
00002  * @file Test_I2C_Compass.c
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 4/29/2015\n Created: 2/28/2015 6:35:33 PM
00006  * @brief This file is for the testing of the I2C Compass code.
00007  */
00008
00009 #ifndef F_CPU
00010     /** @brief The current clock speed for the microcontroller. */
00011     #define F_CPU 16000000UL
00012 #endif
00013
00014 #include <util/delay.h>
00015 #include <stdint.h>
00016 #include <stdio.h>
00017 #include "../Master_Build_Control.h"
00018 #include "../Microcontroller/Communication/I2C_Interface.h"
00019 #include "../Microcontroller/Communication/USART.h"
00020 #include "../Subsystem/IMU/LSM303.h"
00021
00022 void Test_I2C_Compass_Mag_Acc(void)
00023 {
00024     I2C_Interface_Initialize();
00025     USART_Initialize();
00026     LSM303_Initialize();
00027     int16_t Acc[3] = {0, 0, 0};
00028     int16_t Mag[3] = {0, 0, 0};
00029     char Buffer[50] = {0x00};
00030     while(1)
00031     {
00032         LSM303_Read_Mag(Mag);
  
```

```

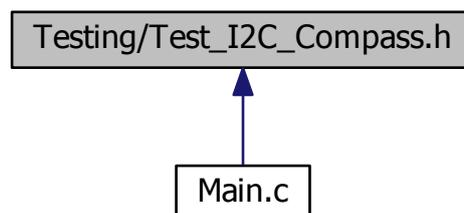
00033     LSM303_Read_Acc(Acc);
00034     sprintf(Buffer,"Acc: %d, %d, %d Mag: %d, %d, %d\r\n", Mag[0], Mag[1], Mag[2], Acc[0], Acc[1], Acc[2
]);
00035     USART_Send_String(Buffer);
00036     _delay_ms(100);
00037 }
00038 }
00039
00040 void Test_I2C_Compass_Heading(void)
00041 {
00042     I2C_Interface_Initialize();
00043     USART_Initialize();
00044     LSM303_Initialize();
00045     int16_t Acc[3] = {0, 0, 0};
00046     int16_t Mag[3] = {0, 0, 0};
00047     char Buffer_Heading[FLOATING_POINT_BUFFER_SIZE] = {0x00};
00048     float Heading = 0;
00049     char Buffer[10] = {0x00};
00050     while(1)
00051     {
00052         LSM303_Read_Mag(Mag);
00053         LSM303_Read_Acc(Acc);
00054         Heading = LSM303_Heading(Acc, Mag);
00055         sprintf(Buffer,"%s\r\n", USART_ftoa(Heading, Buffer_Heading));
00056         USART_Send_String(Buffer);
00057         _delay_ms(100);
00058     }
00059 }
00060
00061 /**
00062  * @brief This function is used to test the compass code.
00063  */
00064 void Test_I2C_Compass(void)
00065 {
00066     #if Compass_Test_Mode == 0
00067         Test_I2C_Compass_Mag_Acc();
00068     #elif Compass_Test_Mode == 1
00069         Test_I2C_Compass_Heading();
00070     #endif
00071 }

```

3.84 Testing/Test_I2C_Compass.h File Reference

This is the header file for the testing of the I2C Compass code.

This graph shows which files directly or indirectly include this file:



Functions

- void [Test_I2C_Compass](#) (void)

This function is used to test the compass code.

3.84.1 Detailed Description

This is the header file for the testing of the I2C Compass code.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 4/29/2015

Created: 2/28/2015 6:35:13 PM

Definition in file [Test_I2C_Compass.h](#).

3.84.2 Function Documentation

3.84.2.1 void Test_I2C_Compass (void)

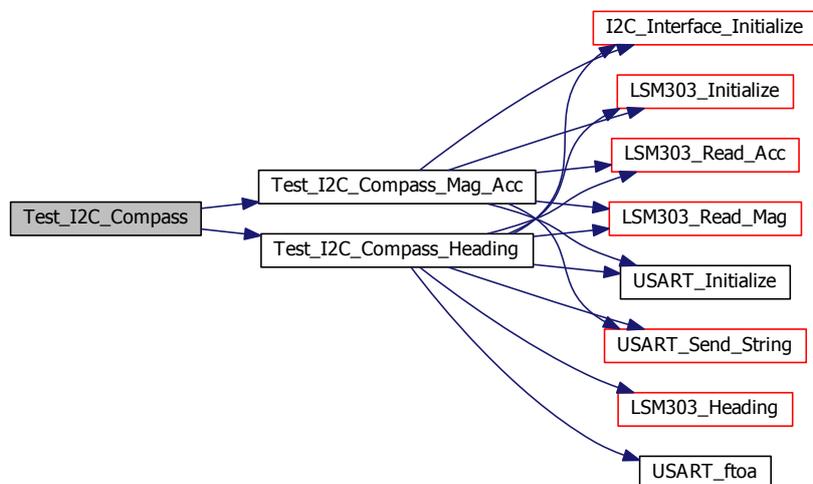
This function is used to test the compass code.

Definition at line 64 of file [Test_I2C_Compass.c](#).

References [Test_I2C_Compass_Heading\(\)](#), and [Test_I2C_Compass_Mag_Acc\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.85 Test_I2C_Compass.h

```

00001 /**
00002  * @file Test_I2C_Compass.h
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 4/29/2015\n Created: 2/28/2015 6:35:13 PM
00006  * @brief This is the header file for the testing of the I2C Compass code.
00007  */
00008
00009 #ifndef TEST_I2C_COMPASS_H_
00010 #define TEST_I2C_COMPASS_H_
00011
00012 void Test_I2C_Compass(void);
00013
00014 #endif /* TEST_I2C_COMPASS_H_ */
  
```

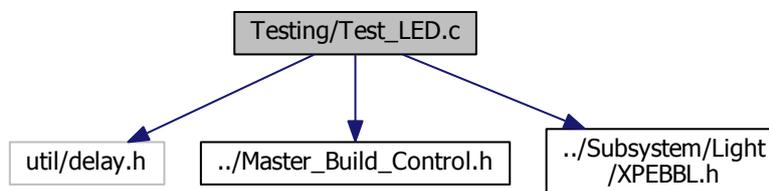
3.86 Testing/Test_LED.c File Reference

This file is for the testing of the LED code.

```

#include <util/delay.h>
#include "../Master_Build_Control.h"
#include "../Subsystem/Light/XPEBBL.h"
  
```

Include dependency graph for Test_LED.c:



Macros

- #define F_CPU 16000000UL

The current clock speed for the microcontroller.

Functions

- void [Test_LED_Front_Toggle](#) (void)
- void [Test_LED_Change_LED](#) (void)
- void [Test_LED](#) (void)

This function is used to test the toggling of the LED code.

3.86.1 Detailed Description

This file is for the testing of the LED code.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 4/29/2015

Created: 2/28/2015 7:31:18 PM

Definition in file [Test_LED.c](#).

3.86.2 Macro Definition Documentation

3.86.2.1 #define F_CPU 16000000UL

The current clock speed for the microcontroller.

Definition at line 11 of file [Test_LED.c](#).

3.86.3 Function Documentation

3.86.3.1 void Test_LED (void)

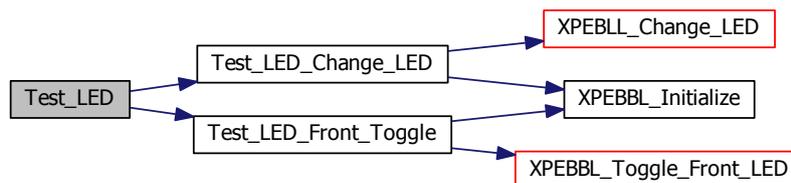
This function is used to test the toggling of the LED code.

Definition at line 43 of file [Test_LED.c](#).

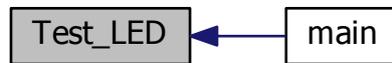
References [Test_LED_Change_LED\(\)](#), and [Test_LED_Front_Toggle\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



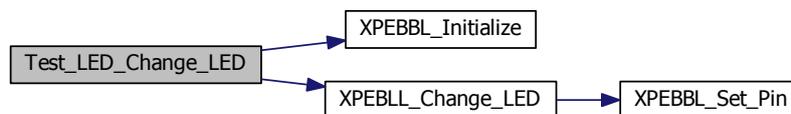
3.86.3.2 void Test_LED_Change_LED (void)

Definition at line 29 of file [Test_LED.c](#).

References [XPEBBL_Initialize\(\)](#), and [XPEBLL_Change_LED\(\)](#).

Referenced by [Test_LED\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



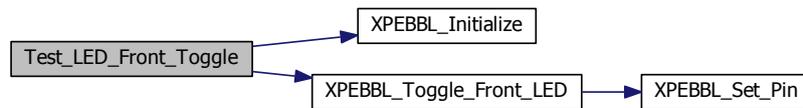
3.86.3.3 void Test_LED_Front_Toggle (void)

Definition at line 18 of file [Test_LED.c](#).

References [XPEBBL_Initialize\(\)](#), and [XPEBBL_Toggle_Front_LED\(\)](#).

Referenced by [Test_LED\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.87 Test_LED.c

```

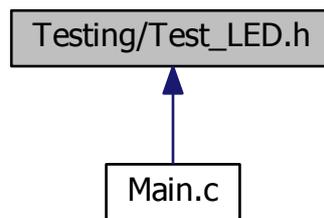
00001 /**
00002  * @file Test_LED.c
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 4/29/2015\n Created: 2/28/2015 7:31:18 PM
00006  * @brief This file is for the testing of the LED code.
00007  */
00008
00009 #ifndef F_CPU
00010   /** @brief The current clock speed for the microcontroller. */
00011   #define F_CPU 16000000UL
00012 #endif
00013
00014 #include <util/delay.h>
00015 #include "../Master_Build_Control.h"
00016 #include "../Subsystem/Light/XPEBBL.h"
00017
00018 void Test_LED_Front_Toggle(void)
00019 {
00020   /* Initialize the LEDs. */
00021   XPEBBL_Initialize();
00022   while(1)
00023   {
00024     XPEBBL_Toggle_Front_LED();
00025     _delay_ms(25);
00026   }
00027 }
00028
00029 void Test_LED_Change_LED(void)
00030 {
00031   /* Initialize the LEDs. */
00032   XPEBBL_Initialize();
00033   while(1)
00034   {
00035     XPEBBL_Change_LED();
00036     _delay_ms(25);
00037   }
00038 }
00039
00040 /**
00041  * @brief This function is used to test the toggling of the LED code.
00042  */
00043 void Test_LED(void)
00044 {
00045   #if LED_Test_Mode == 0
00046     Test_LED_Front_Toggle();
00047   #elif LED_Test_Mode == 1
  
```

```
00048     Test_LED_Change_LED ();
00049     #endif
00050 }
```

3.88 Testing/Test_LED.h File Reference

This is the header file for the testing of the LED code.

This graph shows which files directly or indirectly include this file:



Functions

- void [Test_LED](#) (void)
This function is used to test the toggling of the LED code.

3.88.1 Detailed Description

This is the header file for the testing of the LED code.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 4/29/2015
Created: 2/28/2015 7:31:32 PM

Definition in file [Test_LED.h](#).

3.88.2 Function Documentation

3.88.2.1 void Test_LED (void)

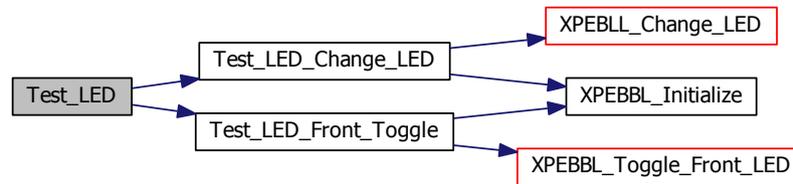
This function is used to test the toggling of the LED code.

Definition at line 43 of file [Test_LED.c](#).

References [Test_LED_Change_LED\(\)](#), and [Test_LED_Front_Toggle\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.89 Test_LED.h

```

00001 /**
00002  * @file Test_LED.h
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 4/29/2015\n Created: 2/28/2015 7:31:32 PM
00006  * @brief This is the header file for the testing of the LED code.
00007  */
00008
00009 #ifndef TEST_LED_H_
00010 #define TEST_LED_H_
00011
00012 void Test_LED(void);
00013
00014 #endif /* TEST_LED_H_ */
  
```

3.90 Testing/Test_Motor_Control.c File Reference

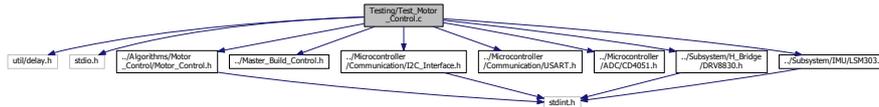
This file is for the testing of the motor control code.

```

#include <util/delay.h>
#include <stdio.h>
#include "../Algorithms/Motor_Control/Motor_Control.h"
#include "../Master_Build_Control.h"
#include "../Microcontroller/Communication/I2C_Interface.h"
#include "../Microcontroller/Communication/USART.h"
#include "../Microcontroller/ADC/CD4051.h"
#include "../Subsystem/H_Bridge/DRV8830.h"
#include "../Subsystem/IMU/LSM303.h"

```

Include dependency graph for Test_Motor_Control.c:



Macros

- `#define F_CPU 16000000UL`
The current clock speed for the microcontroller.
- `#define Motor_Control_Test_Speed 1000`
*Update speeds for the motor control test function.
The rate is equal to (50 ms)*(variable value).*

Functions

- void `Test_Motor_Control_X_Varying` (void)
- void `Test_Motor_Control_Y_Varying` (void)
- void `Test_Motor_Control_Z_Varying` (void)
- void `Test_Motor_Control_X_Constant` (void)
- void `Test_Motor_Control_Y_Constant` (void)
- void `Test_Motor_Control_Z_Constant` (void)
- void `Test_Motor_Control` (void)

This function is used to test the motor control loops for the three motors.

3.90.1 Detailed Description

This file is for the testing of the motor control code.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 4/29/2015

Created: 2/28/2015 6:31:21 PM

Definition in file [Test_Motor_Control.c](#).

3.90.2 Macro Definition Documentation

3.90.2.1 #define F_CPU 16000000UL

The current clock speed for the microcontroller.

Definition at line 11 of file [Test_Motor_Control.c](#).

3.90.2.2 #define Motor_Control_Test_Speed 1000

Update speeds for the motor control test function.

The rate is equal to (50 ms)*(variable value).

Warning

This is value needs to be an integer.

Definition at line 18 of file [Test_Motor_Control.c](#).

Referenced by [Test_Motor_Control_X_Varying\(\)](#), [Test_Motor_Control_Y_Varying\(\)](#), and [Test_Motor_Control_Z_↔ Varying\(\)](#).

3.90.3 Function Documentation

3.90.3.1 void Test_Motor_Control (void)

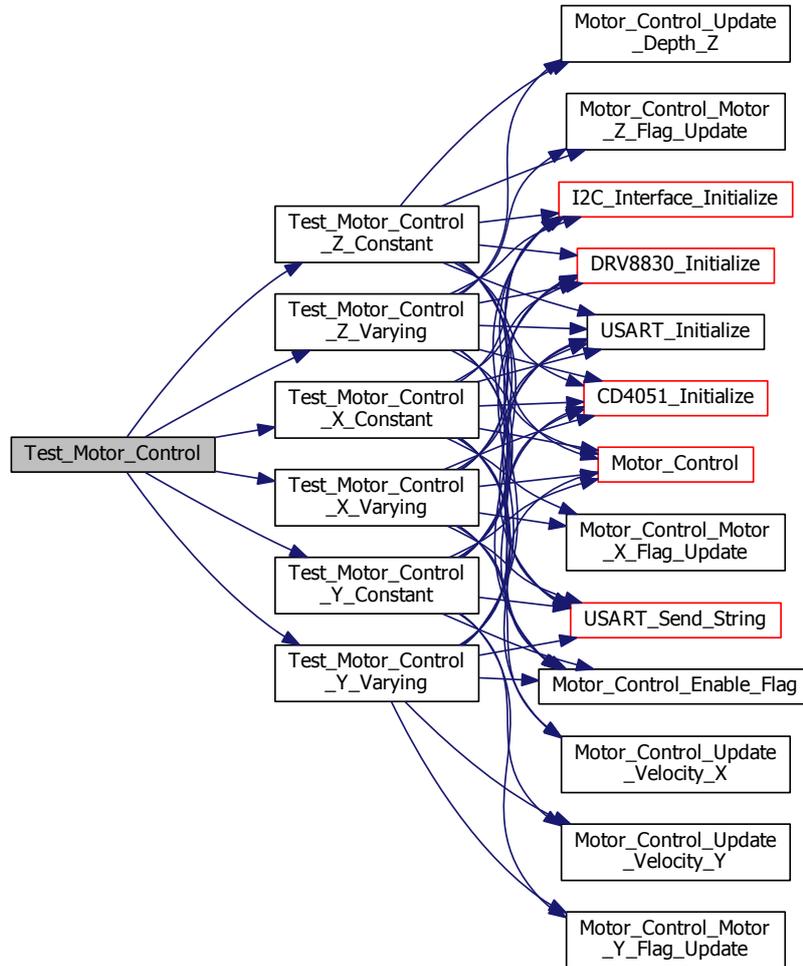
This function is used to test the motor control loops for the three motors.

Definition at line 174 of file [Test_Motor_Control.c](#).

References [Test_Motor_Control_X_Constant\(\)](#), [Test_Motor_Control_X_Varying\(\)](#), [Test_Motor_Control_Y_↔ Constant\(\)](#), [Test_Motor_Control_Y_Varying\(\)](#), [Test_Motor_Control_Z_Constant\(\)](#), and [Test_Motor_Control_Z_↔ Varying\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.90.3.2 void Test_Motor_Control_X_Constant (void)

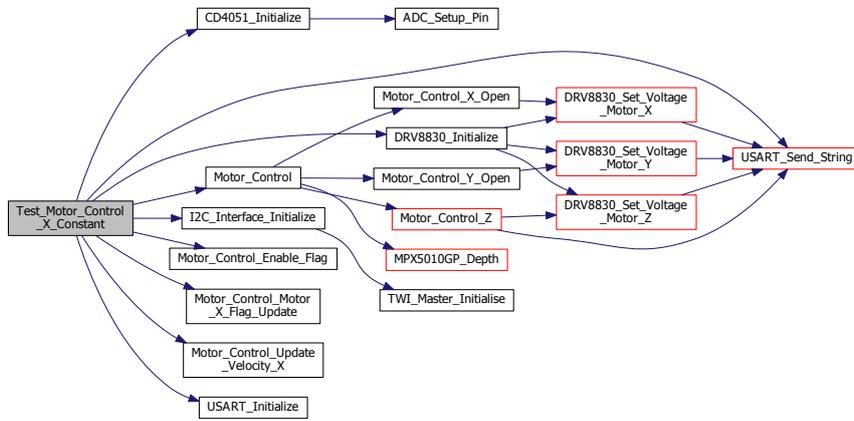
Definition at line 111 of file [Test_Motor_Control.c](#).

References [CD4051_Initialize\(\)](#), [DRV8830_Initialize\(\)](#), [I2C_Interface_Initialize\(\)](#), [Motor_Control\(\)](#), [Motor_Control_Enable_Flag\(\)](#), [Motor_Control_Motor_X_Flag_Update\(\)](#), [Motor_Control_Update_Velocity_X\(\)](#), [USART_Initialize\(\)](#),

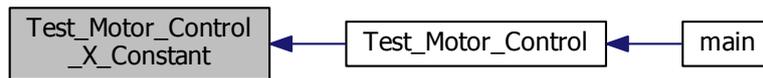
and [USART_Send_String\(\)](#).

Referenced by [Test_Motor_Control\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



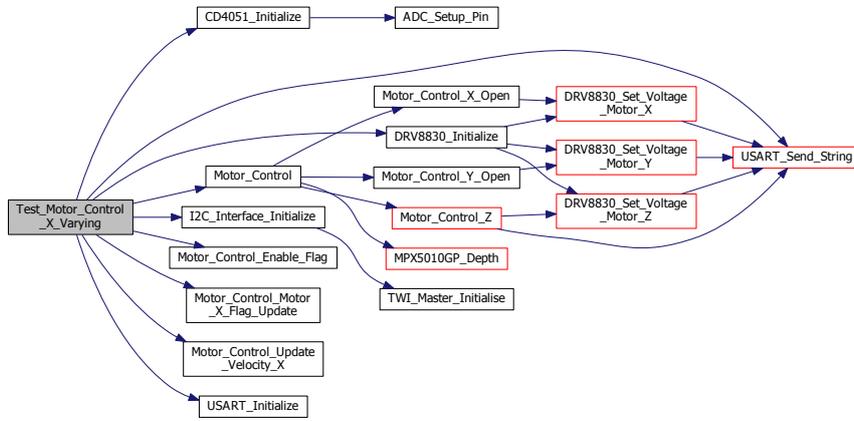
3.90.3.3 void Test_Motor_Control_X_Varying (void)

Definition at line 30 of file [Test_Motor_Control.c](#).

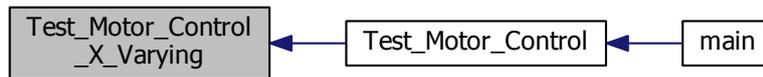
References [CD4051_Initialize\(\)](#), [DRV8830_Initialize\(\)](#), [I2C_Interface_Initialize\(\)](#), [Motor_Control\(\)](#), [Motor_Control_Enable_Flag\(\)](#), [Motor_Control_Motor_X_Flag_Update\(\)](#), [Motor_Control_Test_Speed](#), [Motor_Control_Update_Velocity_X\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_Motor_Control\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



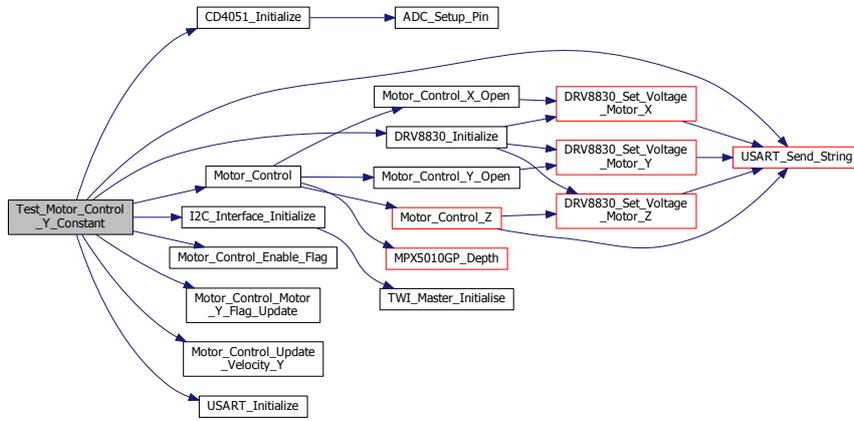
3.90.3.4 void Test_Motor_Control_Y_Constant (void)

Definition at line 131 of file [Test_Motor_Control.c](#).

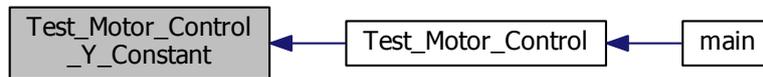
References [CD4051_Initialize\(\)](#), [DRV8830_Initialize\(\)](#), [I2C_Interface_Initialize\(\)](#), [Motor_Control\(\)](#), [Motor_Control_Enable_Flag\(\)](#), [Motor_Control_Motor_Y_Flag_Update\(\)](#), [Motor_Control_Update_Velocity_Y\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_Motor_Control\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



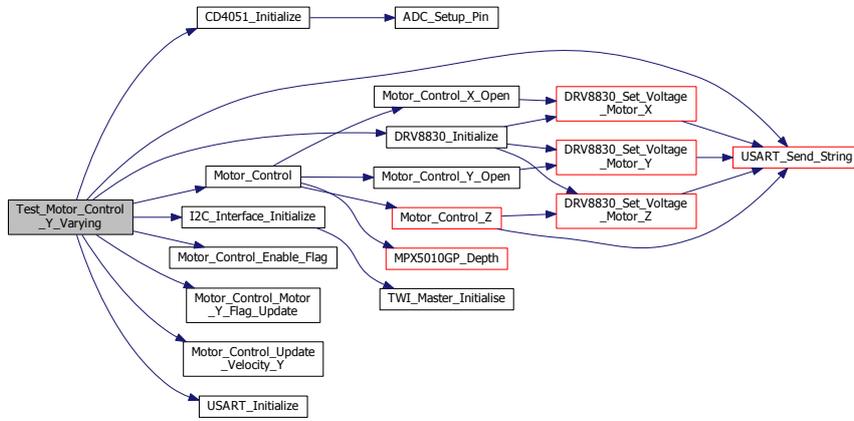
3.90.3.5 void Test_Motor_Control_Y_Varying (void)

Definition at line 57 of file [Test_Motor_Control.c](#).

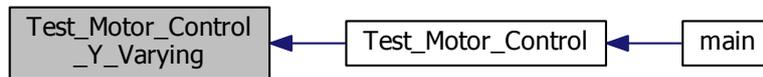
References [CD4051_Initialize\(\)](#), [DRV8830_Initialize\(\)](#), [I2C_Interface_Initialize\(\)](#), [Motor_Control\(\)](#), [Motor_Control_Enable_Flag\(\)](#), [Motor_Control_Motor_Y_Flag_Update\(\)](#), [Motor_Control_Test_Speed](#), [Motor_Control_Update_Velocity_Y\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_Motor_Control\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



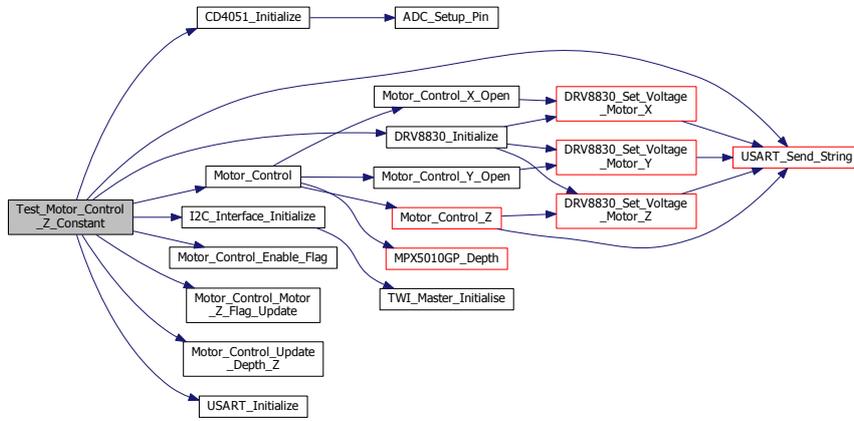
3.90.3.6 void Test_Motor_Control_Z_Constant (void)

Definition at line 151 of file [Test_Motor_Control.c](#).

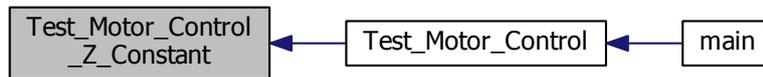
References [CD4051_Initialize\(\)](#), [DRV8830_Initialize\(\)](#), [I2C_Interface_Initialize\(\)](#), [Motor_Control\(\)](#), [Motor_Control_Enable_Flag\(\)](#), [Motor_Control_Motor_Z_Flag_Update\(\)](#), [Motor_Control_Update_Depth_Z\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_Motor_Control\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



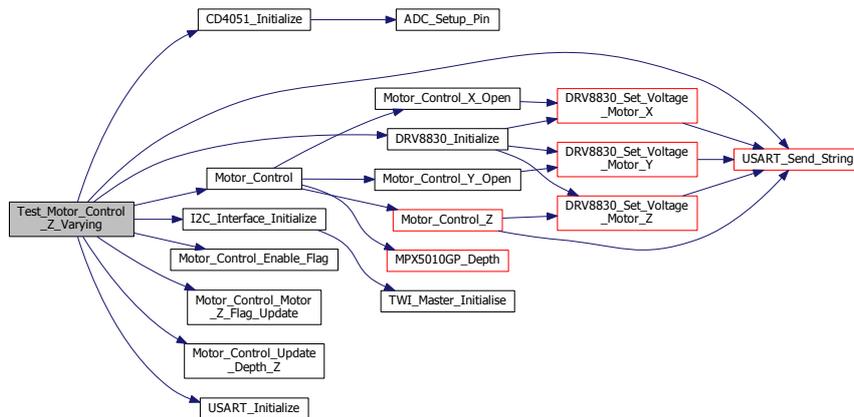
3.90.3.7 void Test_Motor_Control_Z_Varying (void)

Definition at line 84 of file [Test_Motor_Control.c](#).

References [CD4051_Initialize\(\)](#), [DRV8830_Initialize\(\)](#), [I2C_Interface_Initialize\(\)](#), [Motor_Control\(\)](#), [Motor_Control_Enable_Flag\(\)](#), [Motor_Control_Motor_Z_Flag_Update\(\)](#), [Motor_Control_Test_Speed](#), [Motor_Control_Update_Depth_Z\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_Motor_Control\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.91 Test_Motor_Control.c

```

00001 /**
00002  * @file Test_Motor_Control.c
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 4/29/2015\n Created: 2/28/2015 6:31:21 PM
00006  * @brief This file is for the testing of the motor control code.
00007  */
00008
00009 #ifndef F_CPU
00010     /** @brief The current clock speed for the microcontroller. */
00011     #define F_CPU 16000000UL
00012 #endif
00013 /**
00014  * @brief Update speeds for the motor control test function.\n
00015  * The rate is equal to (50 ms)*(variable value).
00016  * @warning This is value needs to be an integer.
00017  */
00018 #define Motor_Control_Test_Speed 1000
00019
00020 #include <util/delay.h>
00021 #include <stdio.h>
00022 #include "../Algorithms/Motor_Control/Motor_Control.h"
00023 #include "../Master_Build_Control.h"
00024 #include "../Microcontroller/Communication/I2C_Interface.h"
00025 #include "../Microcontroller/Communication/USART.h"
00026 #include "../Microcontroller/ADC/CD4051.h"
00027 #include "../Subsystem/H_Bridge/DRV8830.h"
00028 #include "../Subsystem/IMU/LSM303.h"
00029
00030 void Test_Motor_Control_X_Varying(void)
00031 {
00032     I2C_Interface_Initialize();
00033     DRV8830_Initialize();
00034     USART_Initialize();
00035     CD4051_Initialize();
  
```

```

00036     Motor_Control_Motor_X_Flag_Update(1);
00037     char Buffer[50] = {0x00};
00038     while(1)
00039     {
00040         int Motor_Value = -5;
00041         For(int i=0; Motor_Value<=5; i++)
00042         {
00043             Motor_Control_Update_Velocity_X(Motor_Value);
00044             if(i%Motor_Control_Test_Speed==1)
00045             {
00046                 Motor_Value++;
00047             }
00048             Motor_Control();
00049             sprintf(Buffer,"i: %d, Motor Value: %d\r\n", i, Motor_Value);
00050             USART_Send_String(Buffer);
00051             Motor_Control_Enable_Flag();
00052             _delay_ms(50);
00053         }
00054     }
00055 }
00056
00057 void Test_Motor_Control_Y_Varying(void)
00058 {
00059     I2C_Interface_Initialize();
00060     DRV8830_Initialize();
00061     USART_Initialize();
00062     CD4051_Initialize();
00063     Motor_Control_Motor_Y_Flag_Update(1);
00064     char Buffer[50] = {0x00};
00065     while(1)
00066     {
00067         int Motor_Value = -5;
00068         For(int i=0; Motor_Value<=5; i++)
00069         {
00070             Motor_Control_Update_Velocity_Y(Motor_Value);
00071             if(i%Motor_Control_Test_Speed==1)
00072             {
00073                 Motor_Value++;
00074             }
00075             Motor_Control();
00076             sprintf(Buffer,"i: %d, Motor Value: %d\r\n", i, Motor_Value);
00077             USART_Send_String(Buffer);
00078             Motor_Control_Enable_Flag();
00079             _delay_ms(50);
00080         }
00081     }
00082 }
00083
00084 void Test_Motor_Control_Z_Varying(void)
00085 {
00086     I2C_Interface_Initialize();
00087     DRV8830_Initialize();
00088     USART_Initialize();
00089     CD4051_Initialize();
00090     Motor_Control_Motor_Z_Flag_Update(1);
00091     char Buffer[50] = {0x00};
00092     while(1)
00093     {
00094         int Motor_Value = 0;
00095         For(int i=0; Motor_Value<2.5; i++)
00096         {
00097             Motor_Control_Update_Depth_Z(Motor_Value);
00098             if(i%Motor_Control_Test_Speed==1)
00099             {
00100                 Motor_Value++;
00101             }
00102             Motor_Control();
00103             sprintf(Buffer,"i: %d, Motor Value: %d\r\n", i, Motor_Value);
00104             USART_Send_String(Buffer);
00105             Motor_Control_Enable_Flag();
00106             _delay_ms(50);
00107         }
00108     }
00109 }
00110
00111 void Test_Motor_Control_X_Constant(void)
00112 {
00113     I2C_Interface_Initialize();
00114     DRV8830_Initialize();
00115     USART_Initialize();
00116     CD4051_Initialize();
00117     Motor_Control_Motor_X_Flag_Update(1);
00118     char Buffer[50] = {0x00};
00119     int Motor_Value = 0;
00120     while(1)
00121     {
00122         Motor_Control_Update_Velocity_X(Motor_Value);

```

```

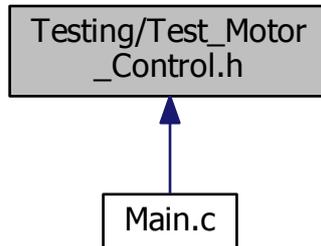
00123     Motor_Control();
00124     sprintf(Buffer, "Motor Value: %d\r\n", Motor_Value);
00125     USART_Send_String(Buffer);
00126     Motor_Control_Enable_Flag();
00127     _delay_ms(50);
00128 }
00129 }
00130
00131 void Test_Motor_Control_Y_Constant(void)
00132 {
00133     I2C_Interface_Initialize();
00134     DRV8830_Initialize();
00135     USART_Initialize();
00136     CD4051_Initialize();
00137     Motor_Control_Motor_Y_Flag_Update(1);
00138     char Buffer[50] = {0x00};
00139     int Motor_Value = 0;
00140     while(1)
00141     {
00142         Motor_Control_Update_Velocity_Y(Motor_Value);
00143         Motor_Control();
00144         sprintf(Buffer, "Motor Value: %d\r\n", Motor_Value);
00145         USART_Send_String(Buffer);
00146         Motor_Control_Enable_Flag();
00147         _delay_ms(50);
00148     }
00149 }
00150
00151 void Test_Motor_Control_Z_Constant(void)
00152 {
00153     I2C_Interface_Initialize();
00154     DRV8830_Initialize();
00155     USART_Initialize();
00156     CD4051_Initialize();
00157     Motor_Control_Motor_Z_Flag_Update(1);
00158     char Buffer[50] = {0x00};
00159     int Motor_Value = 1;
00160     while(1)
00161     {
00162         Motor_Control_Update_Depth_Z(Motor_Value);
00163         Motor_Control();
00164         sprintf(Buffer, "Motor Value: %d\r\n", Motor_Value);
00165         USART_Send_String(Buffer);
00166         Motor_Control_Enable_Flag();
00167         _delay_ms(50);
00168     }
00169 }
00170
00171 /**
00172  * @brief This function is used to test the motor control loops for the three motors.
00173  */
00174 void Test_Motor_Control(void)
00175 {
00176     #if Motor_Control_Test_Mode == 0
00177         Test_Motor_Control_X_Varying();
00178     #elif Motor_Control_Test_Mode == 1
00179         Test_Motor_Control_Y_Varying();
00180     #elif Motor_Control_Test_Mode == 2
00181         Test_Motor_Control_Z_Varying();
00182     #elif Motor_Control_Test_Mode == 3
00183         Test_Motor_Control_X_Constant();
00184     #elif Motor_Control_Test_Mode == 4
00185         Test_Motor_Control_Y_Constant();
00186     #elif Motor_Control_Test_Mode == 5
00187         Test_Motor_Control_Z_Constant();
00188     #endif
00189 }

```

3.92 Testing/Test_Motor_Control.h File Reference

This file is the header file for the testing of the motor control code.

This graph shows which files directly or indirectly include this file:



Functions

- void [Test_Motor_Control](#) (void)

This function is used to test the motor control loops for the three motors.

3.92.1 Detailed Description

This file is the header file for the testing of the motor control code.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 4/29/2015

Created: 2/28/2015 6:31:42 PM

Definition in file [Test_Motor_Control.h](#).

3.92.2 Function Documentation

3.92.2.1 void Test_Motor_Control (void)

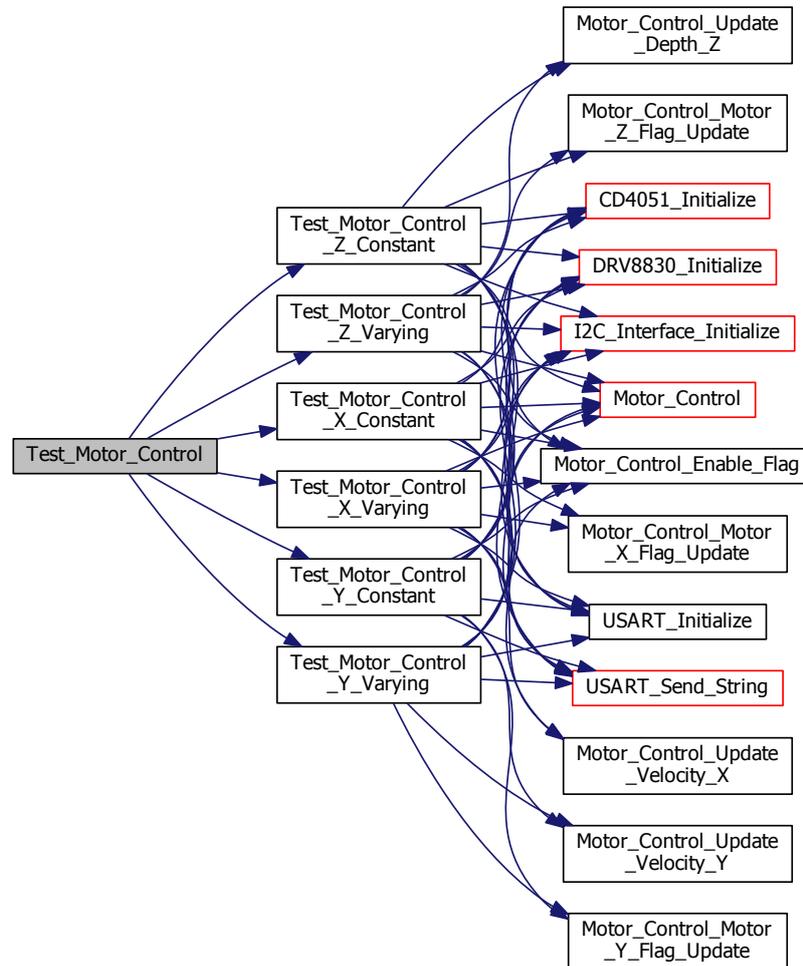
This function is used to test the motor control loops for the three motors.

Definition at line 174 of file [Test_Motor_Control.c](#).

References [Test_Motor_Control_X_Constant\(\)](#), [Test_Motor_Control_X_Varying\(\)](#), [Test_Motor_Control_Y_↔Constant\(\)](#), [Test_Motor_Control_Y_Varying\(\)](#), [Test_Motor_Control_Z_Constant\(\)](#), and [Test_Motor_Control_Z_↔Varying\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.93 Test_Motor_Control.h

```
00001 /**
00002  * @file Test_Motor_Control.h
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 4/29/2015\n Created: 2/28/2015 6:31:42 PM
```

```

00006  * @brief This file is the header file for the testing of the motor control code.
00007  */
00008
00009 #ifndef TEST_MOTOR_CONTROL_H_
00010 #define TEST_MOTOR_CONTROL_H_
00011
00012 void Test_Motor_Control(void);
00013
00014 #endif /* TEST_MOTOR_CONTROL_H_ */

```

3.94 Testing/Test_Multiplexer.c File Reference

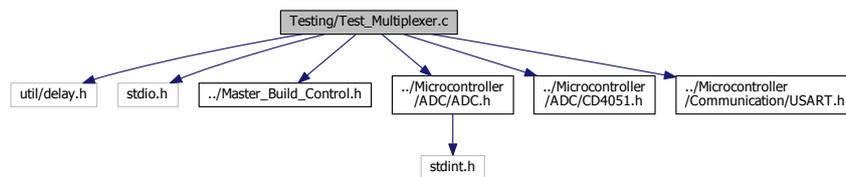
This file is for the testing of the multiplexer code.

```

#include <util/delay.h>
#include <stdio.h>
#include "../Master_Build_Control.h"
#include "../Microcontroller/ADC/ADC.h"
#include "../Microcontroller/ADC/CD4051.h"
#include "../Microcontroller/Communication/USART.h"

```

Include dependency graph for Test_Multiplexer.c:



Macros

- #define `F_CPU` 16000000UL
The current clock speed for the microcontroller.

Functions

- void `Test_Multiplexer_Single_Pin` (void)
- void `Test_Multiplexer_Multiple_Pins` (void)
- void `Test_Multiplexer_Voltage_Conversion` ()
- void `Test_Multiplexer_Voltage_Multiple_Conversions` ()
- void `Test_Multiplexer` (void)

This function is used to test the multiplexer ADC code.

3.94.1 Detailed Description

This file is for the testing of the multiplexer code.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 4/29/2015

Created: 2/28/2015 7:29:55 PM

Definition in file [Test_Multiplexer.c](#).

3.94.2 Macro Definition Documentation

3.94.2.1 #define F_CPU 16000000UL

The current clock speed for the microcontroller.

Definition at line 11 of file [Test_Multiplexer.c](#).

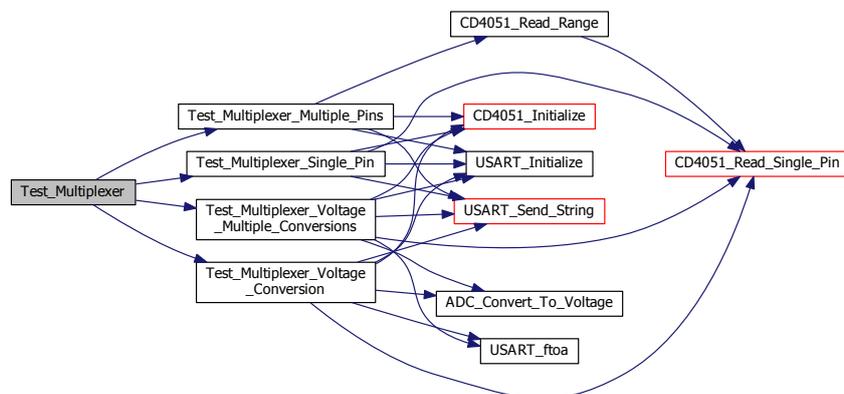
3.94.3 Function Documentation

3.94.3.1 void Test_Multiplexer (void)

This function is used to test the multiplexer ADC code.

Definition at line 120 of file [Test_Multiplexer.c](#).References [Test_Multiplexer_Multiple_Pins\(\)](#), [Test_Multiplexer_Single_Pin\(\)](#), [Test_Multiplexer_Voltage_Conversion\(\)](#), and [Test_Multiplexer_Voltage_Multiple_Conversions\(\)](#).Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



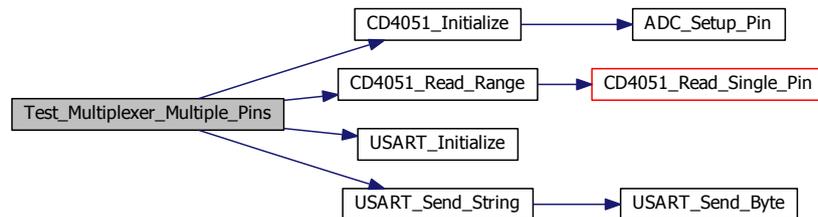
3.94.3.2 void Test_Multiplexer_Multiple_Pins (void)

Definition at line 42 of file [Test_Multiplexer.c](#).

References [CD4051_Initialize\(\)](#), [CD4051_Read_Range\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_Multiplexer\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



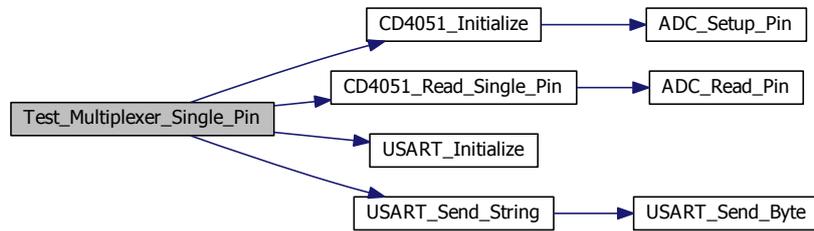
3.94.3.3 void Test_Multiplexer_Single_Pin (void)

Definition at line 21 of file [Test_Multiplexer.c](#).

References [CD4051_Initialize\(\)](#), [CD4051_Read_Single_Pin\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_Multiplexer\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



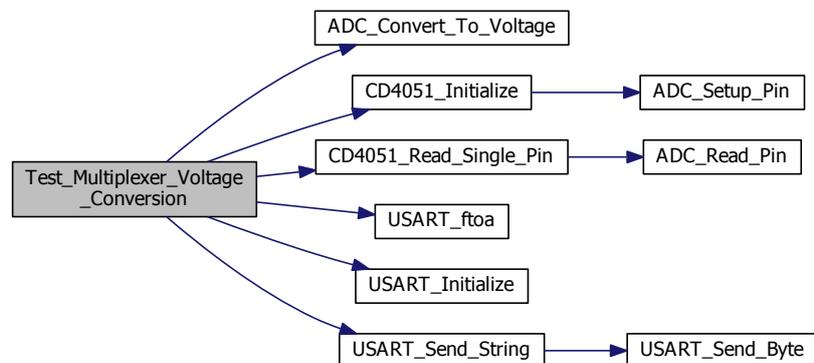
3.94.3.4 void Test_Multiplexer_Voltage_Conversion ()

Definition at line 63 of file [Test_Multiplexer.c](#).

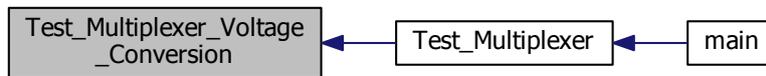
References [ADC_Convert_To_Voltage\(\)](#), [CD4051_Initialize\(\)](#), [CD4051_Read_Single_Pin\(\)](#), [FLOATING_POINT_BUFFER_SIZE](#), [USART_ftoa\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_Multiplexer\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



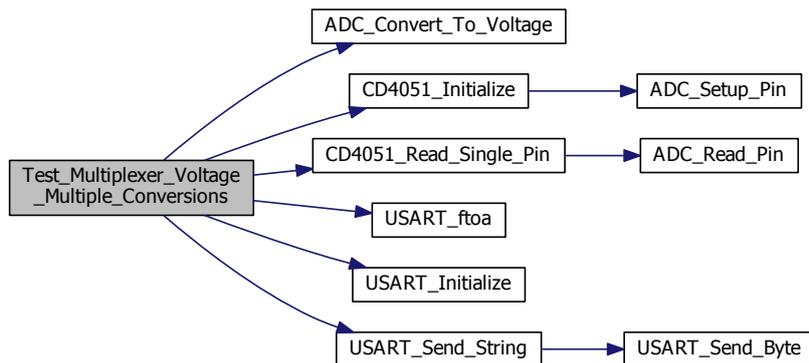
3.94.3.5 void Test_Multiplexer_Voltage_Multiple_Conversions ()

Definition at line 88 of file [Test_Multiplexer.c](#).

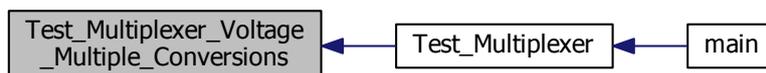
References [ADC_Convert_To_Voltage\(\)](#), [CD4051_Initialize\(\)](#), [CD4051_Read_Single_Pin\(\)](#), [FLOATING_POINT_BUFFER_SIZE](#), [USART_ftoa\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_Multiplexer\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.95 Test_Multiplexer.c

```

00001 /**
00002  * @file Test_Multiplexer.c
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
  
```

```

00005  * @date Last Updated: 4/29/2015\n Created: 2/28/2015 7:29:55 PM
00006  * @brief This file is for the testing of the multiplexer code.
00007  */
00008
00009 #ifndef F_CPU
00010     /** @brief The current clock speed for the microcontroller. */
00011     #define F_CPU 16000000UL
00012 #endif
00013
00014 #include <util/delay.h>
00015 #include <stdio.h>
00016 #include "../Master_Build_Control.h"
00017 #include "../Microcontroller/ADC/ADC.h"
00018 #include "../Microcontroller/ADC/CD4051.h"
00019 #include "../Microcontroller/Communication/USART.h"
00020
00021 void Test_Multiplexer_Single_Pin(void)
00022 {
00023     /* Initialize the multiplexer. */
00024     CD4051_Initialize();
00025     /* Initialize USART communication. */
00026     USART_Initialize();
00027     /* Initialize the temporary variables. */
00028     int ADC_Value_Pin = 0;
00029     char Buffer[20] = {0x00};
00030     while(1)
00031     {
00032         /* Read the ADC value. */
00033         ADC_Value_Pin = CD4051_Read_Single_Pin(0x0);
00034         /* Print the ADC value. */
00035         sprintf(Buffer,"%d\r\n", ADC_Value_Pin);
00036         USART_Send_String(Buffer);
00037         /* Delay between the last check. */
00038         _delay_ms(50);
00039     }
00040 }
00041
00042 void Test_Multiplexer_Multiple_Pins(void)
00043 {
00044     /* Initialize the multiplexer. */
00045     CD4051_Initialize();
00046     /* Initialize USART communication. */
00047     USART_Initialize();
00048     /* Initialize the temporary variables. */
00049     int ADC_Values[] = {0,0,0,0,0,0,0,0};
00050     char Buffer[100] = {0x00};
00051     while(1)
00052     {
00053         /* Read the range of ADC values. */
00054         CD4051_Read_Range(ADC_Values, 0x0, 0x7);
00055         /* Print the range of ADC values. */
00056         sprintf(Buffer,"%d, %d, %d, %d, %d, %d, %d, %d\r\n", ADC_Values[0], ADC_Values[1], ADC_Values[2],
ADC_Values[3], ADC_Values[4], ADC_Values[5], ADC_Values[6], ADC_Values[7]);
00057         USART_Send_String(Buffer);
00058         /* Delay between the last check. */
00059         _delay_ms(50);
00060     }
00061 }
00062
00063 void Test_Multiplexer_Voltage_Conversion()
00064 {
00065     /* Initialize the multiplexer. */
00066     CD4051_Initialize();
00067     /* Initialize USART communication. */
00068     USART_Initialize();
00069     /* Initialize the temporary variables. */
00070     int ADC_Value = 0;
00071     float Voltage_Value = 0;
00072     char Buffer_Voltage_Value[FLOATING_POINT_BUFFER_SIZE] = {0x00};
00073     char Buffer[20] = {0x00};
00074     while(1)
00075     {
00076         /* Read the ADC value. */
00077         ADC_Value = CD4051_Read_Single_Pin(0x0);
00078         /* Convert the ADC value to a voltage. */
00079         Voltage_Value = ADC_Convert_To_Voltage(ADC_Value);
00080         /* Print the ADC and voltage value. */
00081         sprintf(Buffer,"%d, %s V\r\n", ADC_Value, USART_ftoa(Voltage_Value, Buffer_Voltage_Value)
);
00082         USART_Send_String(Buffer);
00083         /* Delay between the last check. */
00084         _delay_ms(50);
00085     }
00086 }
00087
00088 void Test_Multiplexer_Voltage_Multiple_Conversions()
00089 {

```

```

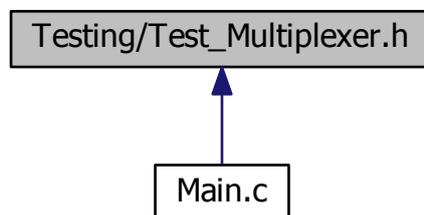
00090     /* Initialize the multiplexer. */
00091     CD4051_Initialize();
00092     /* Initialize USART communication. */
00093     USART_Initialize();
00094     /* Initialize the temporary variables. */
00095     int ADC_Value = 0;
00096     float Voltage_Value = 0;
00097     char Buffer_Voltage_Value[FLOATING_POINT_BUFFER_SIZE] = {0x00};
00098     char Buffer[20] = {0x00};
00099     while(1)
00100     {
00101         /* Loop through the indexes for the multiplexer. */
00102         for(int i = 0; i < 7; i++)
00103         {
00104             /* Read the ADC value. */
00105             ADC_Value = CD4051_Read_Single_Pin(i);
00106             /* Convert the ADC value to a voltage. */
00107             Voltage_Value = ADC_Convert_To_Voltage(ADC_Value);
00108             /* Print the ADC and voltage value. */
00109             sprintf(Buffer, "%d, %s V\r\n", ADC_Value, USART_ftoa(Voltage_Value,
Buffer_Voltage_Value));
00110             USART_Send_String(Buffer);
00111         }
00112         /* Delay between the last check. */
00113         _delay_ms(50);
00114     }
00115 }
00116
00117 /**
00118  * @brief This function is used to test the multiplexer ADC code.
00119  */
00120 void Test_Multiplexer(void)
00121 {
00122     #if Multiplexer_Test_Mode == 0
00123         Test_Multiplexer_Single_Pin();
00124     #elif Multiplexer_Test_Mode == 1
00125         Test_Multiplexer_Multiple_Pins();
00126     #elif Multiplexer_Test_Mode == 2
00127         Test_Multiplexer_Voltage_Conversion();
00128     #elif Multiplexer_Test_Mode == 3
00129         Test_Multiplexer_Voltage_Multiple_Conversions();
00130     #endif
00131 }

```

3.96 Testing/Test_Multiplexer.h File Reference

This file is the header file for the testing of the battery code.

This graph shows which files directly or indirectly include this file:



Functions

- void [Test_Multiplexer](#) (void)

This function is used to test the multiplexer ADC code.

3.96.1 Detailed Description

This file is the header file for the testing of the battery code.

This file is the header file for the testing of the multiplexer code.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 4/29/2015

Created: 2/28/2015 6:39:10 PM

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 4/29/2015

Created: 2/28/2015 7:30:52 PM

Definition in file [Test_Multiplexer.h](#).

3.96.2 Function Documentation

3.96.2.1 void Test_Multiplexer (void)

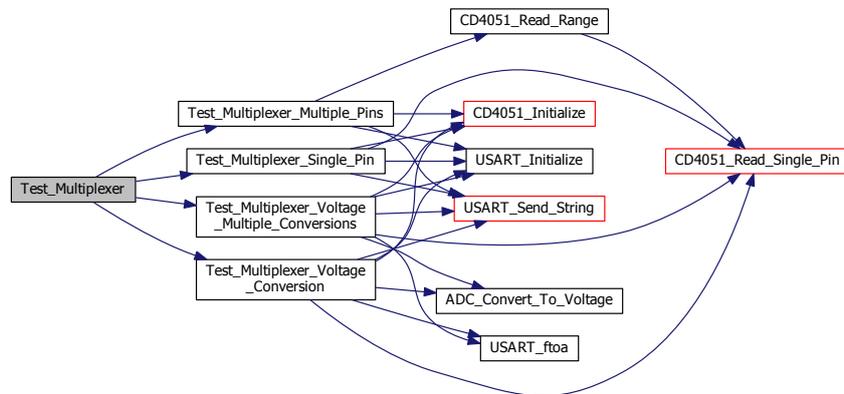
This function is used to test the multiplexer ADC code.

Definition at line 120 of file [Test_Multiplexer.c](#).

References [Test_Multiplexer_Multiple_Pins\(\)](#), [Test_Multiplexer_Single_Pin\(\)](#), [Test_Multiplexer_Voltage_Conversion\(\)](#), and [Test_Multiplexer_Voltage_Multiple_Conversions\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.97 Test_Multiplexer.h

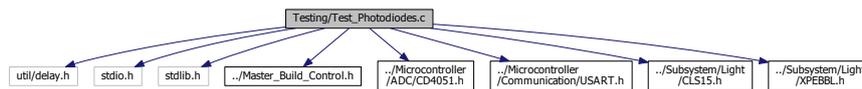
```

00001 /**
00002  * @file Test_Multiplexer.h
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 4/29/2015\n Created: 2/28/2015 7:30:52 PM
00006  * @brief This file is the header file for the testing of the multiplexer code.
00007  */
00008
00009 #ifndef TEST_MULTIPLEXER_H_
00010 #define TEST_MULTIPLEXER_H_
00011
00012 void Test_Multiplexer(void);
00013
00014 #endif /* TEST_MULTIPLEXER_H_ */
  
```

3.98 Testing/Test_Photodiodes.c File Reference

This file is for the testing of the photodiode code.

```
#include <util/delay.h>
#include <stdio.h>
#include <stdlib.h>
#include "../Master_Build_Control.h"
#include "../Microcontroller/ADC/CD4051.h"
#include "../Microcontroller/Communication/USART.h"
#include "../Subsystem/Light/CLS15.h"
#include "../Subsystem/Light/XPEBBL.h"
Include dependency graph for Test_Photodiodes.c:
```



Macros

- `#define F_CPU 16000000UL`
The current clock speed for the microcontroller.

Functions

- void [Test_Photodiodes_Print_Front](#) (void)
- void [Test_Photodiodes_Print_Left](#) (void)
- void [Test_Photodiodes_Print_Right](#) (void)
- void [Test_Photodiodes_Print_Rear](#) (void)
- void [Test_Photodiodes_Front](#) (void)
- void [Test_Photodiodes_Left](#) (void)
- void [Test_Photodiodes_Right](#) (void)
- void [Test_Photodiodes_Rear](#) (void)
- void [Test_Photodiodes_All](#) (void)
- void [Test_Photodiodes_All_With_LED](#) (void)
- void [Test_Photodiodes](#) (void)

This function is used to test the reading of the photodiodes code.

3.98.1 Detailed Description

This file is for the testing of the photodiode code.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 4/29/2015
Created: 2/28/2015 7:28:51 PM

Definition in file [Test_Photodiodes.c](#).

3.98.2 Macro Definition Documentation

3.98.2.1 #define F_CPU 16000000UL

The current clock speed for the microcontroller.

Definition at line 11 of file [Test_Photodiodes.c](#).

3.98.3 Function Documentation

3.98.3.1 void Test_Photodiodes (void)

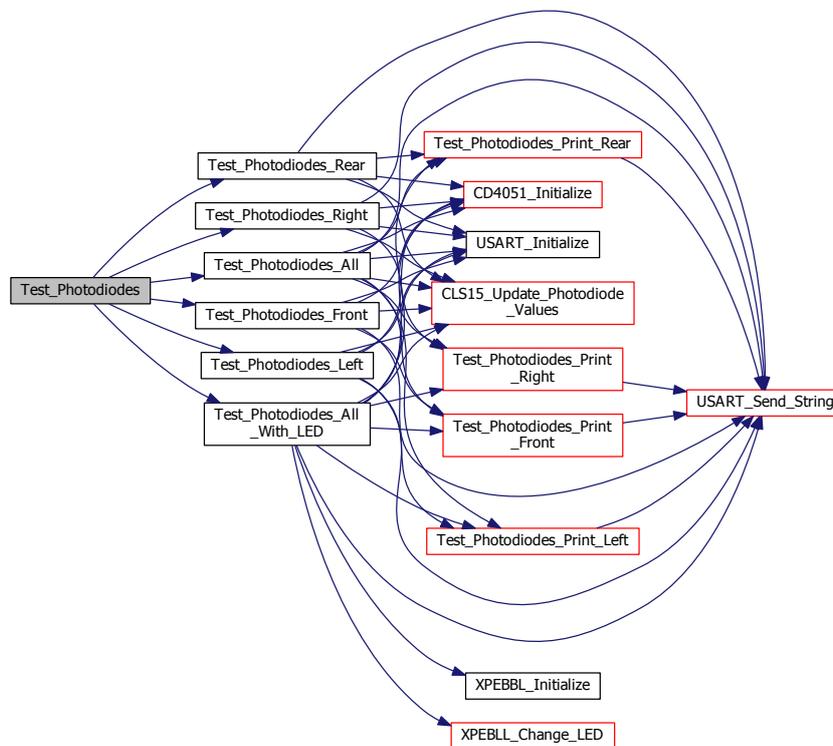
This function is used to test the reading of the photodiodes code.

Definition at line 204 of file [Test_Photodiodes.c](#).

References [Test_Photodiodes_All\(\)](#), [Test_Photodiodes_All_With_LED\(\)](#), [Test_Photodiodes_Front\(\)](#), [Test_Photodiodes_Left\(\)](#), [Test_Photodiodes_Rear\(\)](#), and [Test_Photodiodes_Right\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



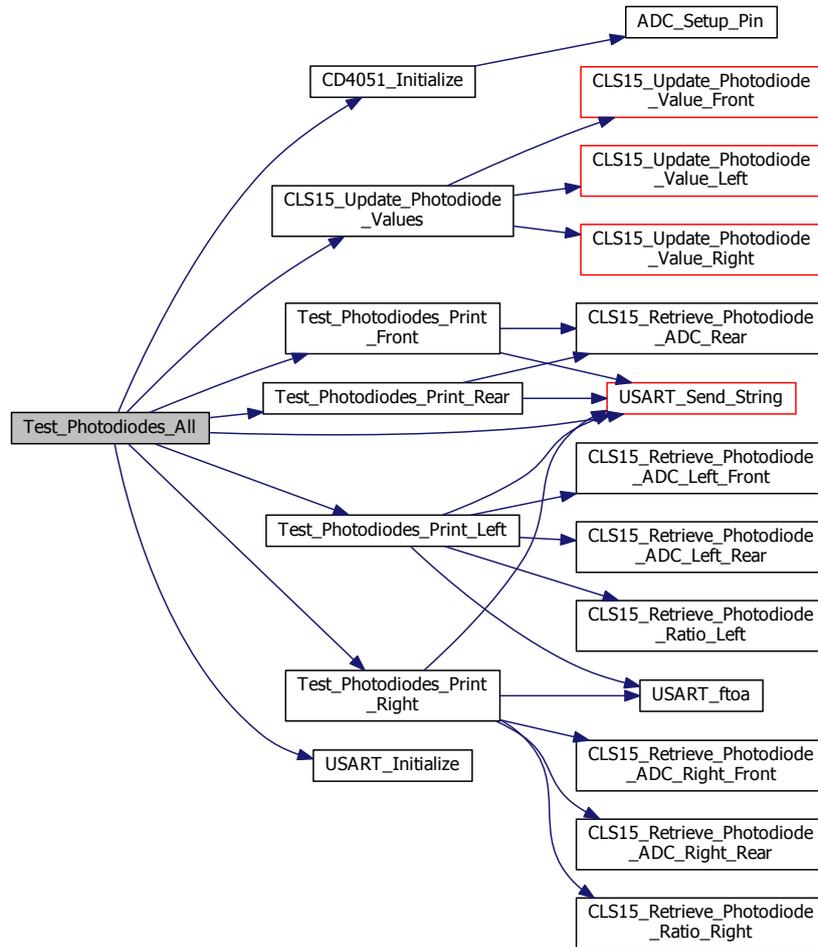
3.98.3.2 void Test_Photodiodes_All (void)

Definition at line 149 of file [Test_Photodiodes.c](#).

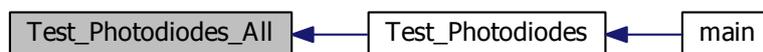
References [CD4051_Initialize\(\)](#), [CLS15_Update_Phodiode_Values\(\)](#), [Test_Photodiodes_Print_Front\(\)](#), [Test_Phodiodes_Print_Left\(\)](#), [Test_Phodiodes_Print_Rear\(\)](#), [Test_Phodiodes_Print_Right\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_Phodiodes\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



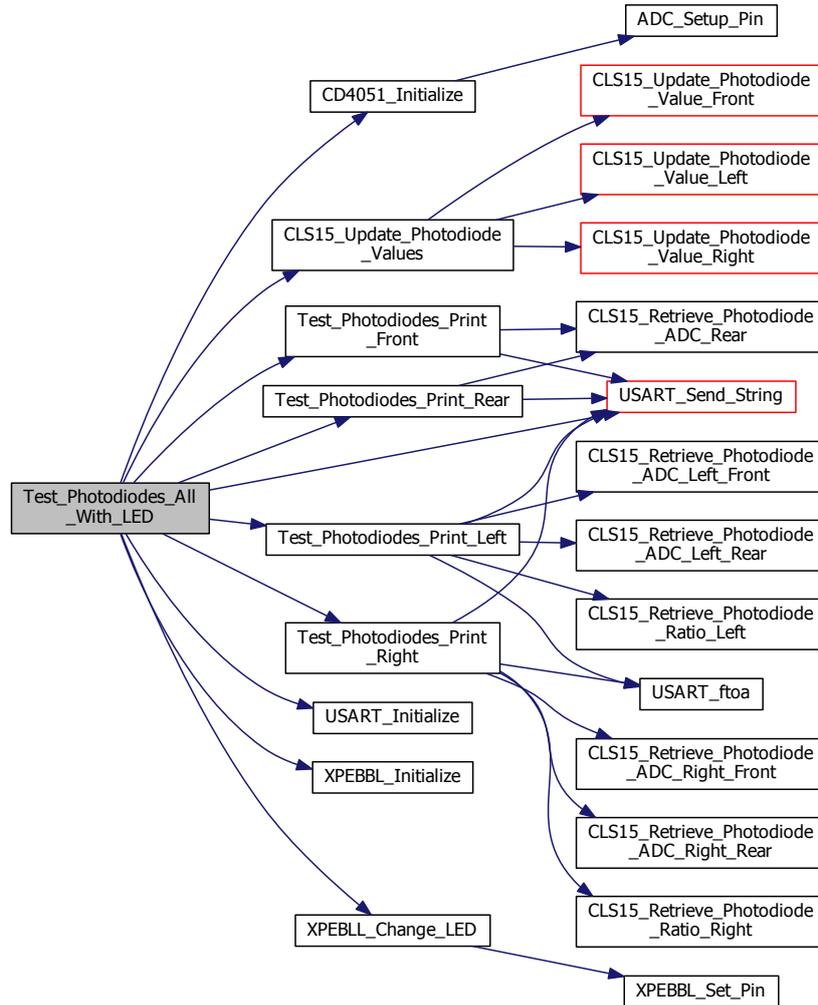
3.98.3.3 void Test_Phodiodes_All_With_LED (void)

Definition at line 174 of file [Test_Phodiodes.c](#).

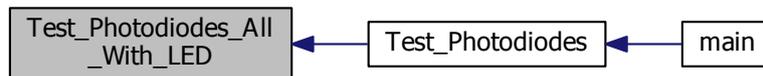
References [CD4051_Initialize\(\)](#), [CLS15_Update_Phodiode_Values\(\)](#), [Test_Phodiodes_Print_Front\(\)](#), [Test_Phodiodes_Print_Left\(\)](#), [Test_Phodiodes_Print_Rear\(\)](#), [Test_Phodiodes_Print_Right\(\)](#), [USART_Initialize\(\)](#), [USART_Send_String\(\)](#), [XPEBLL_Initialize\(\)](#), and [XPEBLL_Change_LED\(\)](#).

Referenced by [Test_Phodiodes\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



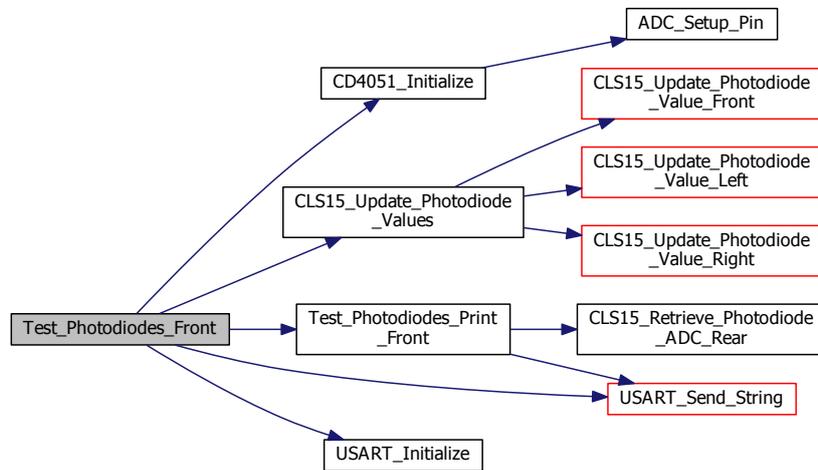
3.98.3.4 void Test_Photodiodes_Front (void)

Definition at line 73 of file [Test_Photodiodes.c](#).

References [CD4051_Initialize\(\)](#), [CLS15_Update_Photodiode_Values\(\)](#), [Test_Photodiodes_Print_Front\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_Phodiodes\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



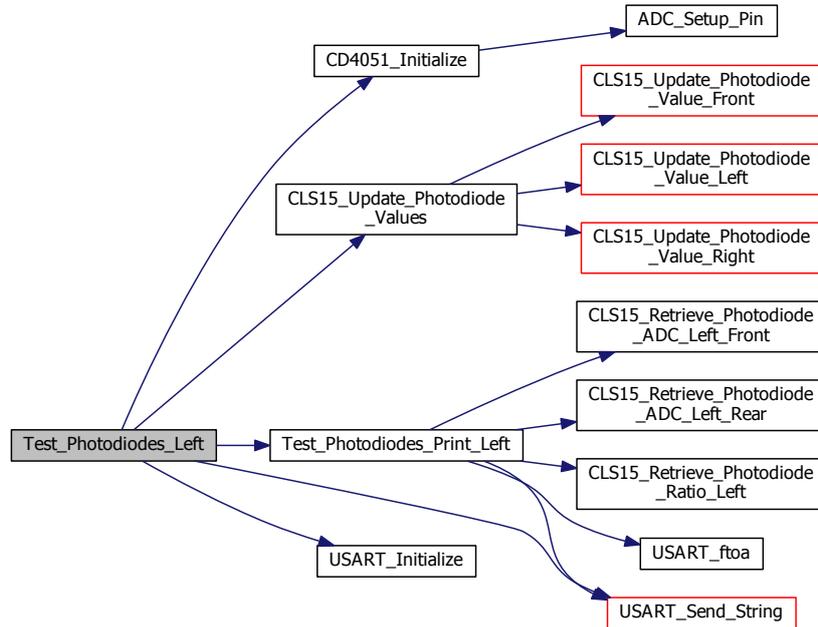
3.98.3.5 void Test_Phodiodes_Left (void)

Definition at line 92 of file [Test_Phodiodes.c](#).

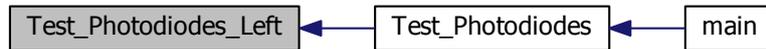
References [CD4051_Initialize\(\)](#), [CLS15_Update_Phodiode_Values\(\)](#), [Test_Phodiodes_Print_Left\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_Phodiodes\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



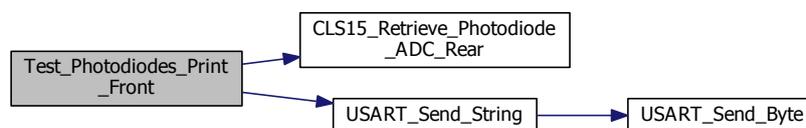
3.98.3.6 void Test_Photodiodes_Print_Front (void)

Definition at line 23 of file Test_Photodiodes.c.

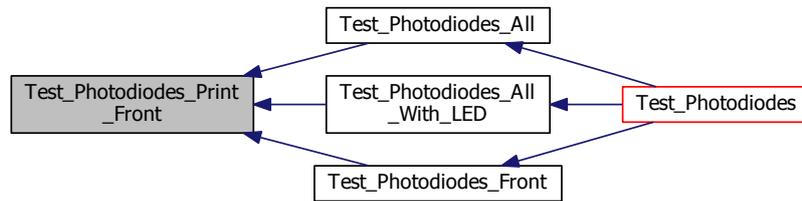
References CLS15_Retrieve_Photodiode_ADC_Rear(), and USART_Send_String().

Referenced by Test_Photodiodes_All(), Test_Photodiodes_All_With_LED(), and Test_Photodiodes_Front().

Here is the call graph for this function:



Here is the caller graph for this function:



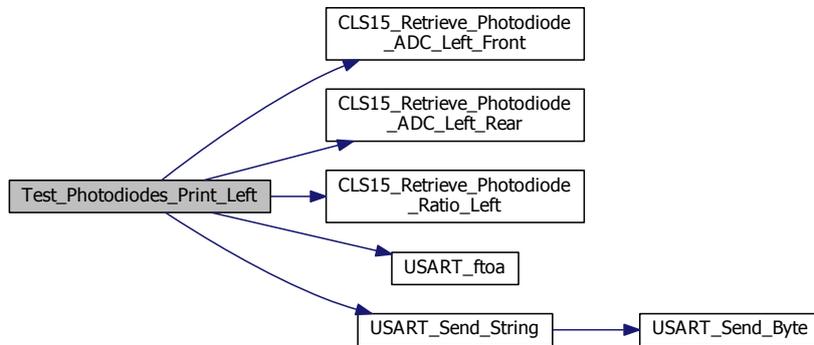
3.98.3.7 void Test_Phodiodes_Print_Left (void)

Definition at line 34 of file Test_Phodiodes.c.

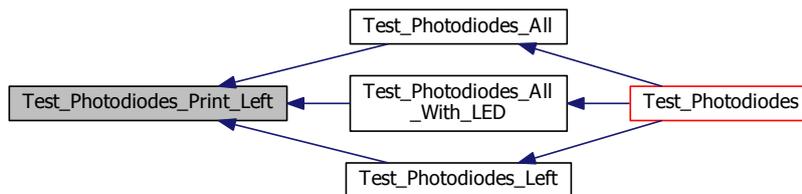
References [CLS15_Retrieve_Phodiode_ADC_Left_Front\(\)](#), [CLS15_Retrieve_Phodiode_ADC_Left_Rear\(\)](#), [CLS15_Retrieve_Phodiode_Ratio_Left\(\)](#), [FLOATING_POINT_BUFFER_SIZE](#), [USART_ftoa\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_Phodiodes_All\(\)](#), [Test_Phodiodes_All_With_LED\(\)](#), and [Test_Phodiodes_Left\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



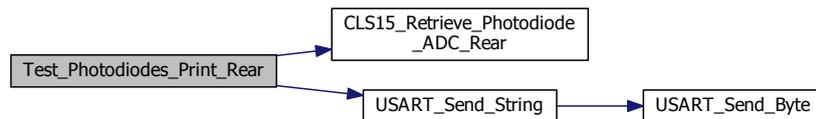
3.98.3.8 void Test_Photodiodes_Print_Rear (void)

Definition at line 62 of file [Test_Photodiodes.c](#).

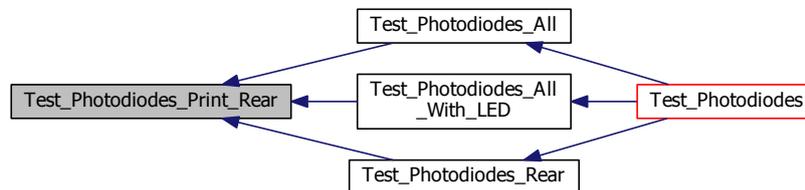
References [CLS15_Retrieve_Photodiode_ADC_Rear\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_Photodiodes_All\(\)](#), [Test_Photodiodes_All_With_LED\(\)](#), and [Test_Photodiodes_Rear\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



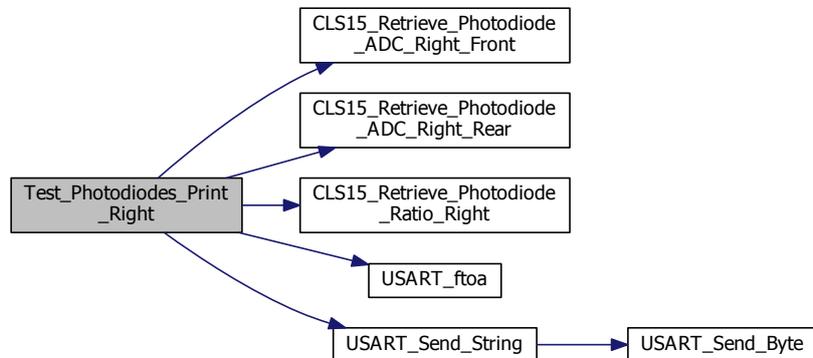
3.98.3.9 void Test_Photodiodes_Print_Right (void)

Definition at line 48 of file [Test_Photodiodes.c](#).

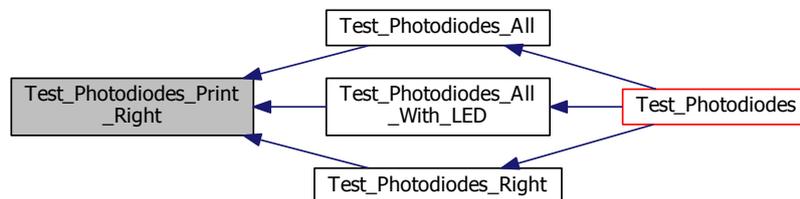
References [CLS15_Retrieve_Photodiode_ADC_Right_Front\(\)](#), [CLS15_Retrieve_Photodiode_ADC_Right_Rear\(\)](#), [CLS15_Retrieve_Photodiode_Ratio_Right\(\)](#), [FLOATING_POINT_BUFFER_SIZE](#), [USART_ftoa\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_Photodiodes_All\(\)](#), [Test_Photodiodes_All_With_LED\(\)](#), and [Test_Photodiodes_Right\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



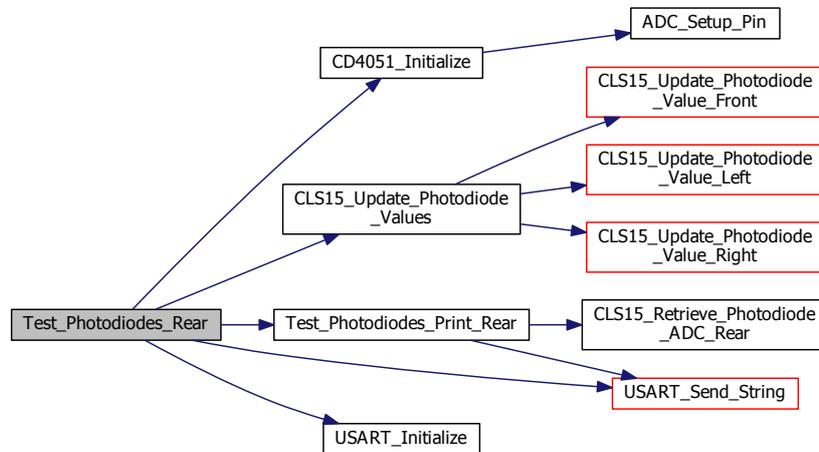
3.98.3.10 void Test_Photodiodes_Rear (void)

Definition at line 130 of file [Test_Photodiodes.c](#).

References [CD4051_Initialize\(\)](#), [CLS15_Update_Photodiode_Values\(\)](#), [Test_Photodiodes_Print_Rear\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_Photodiodes\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



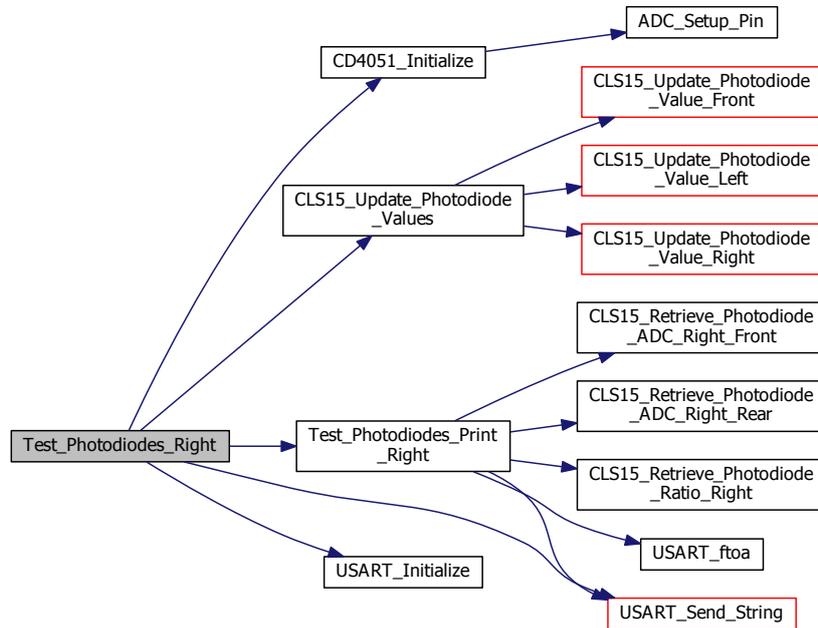
3.98.3.11 void Test_Photodiodes_Right (void)

Definition at line 111 of file [Test_Photodiodes.c](#).

References [CD4051_Initialize\(\)](#), [CLS15_Update_Photodiode_Values\(\)](#), [Test_Photodiodes_Print_Right\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [Test_Photodiodes\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.99 Test_Phodiodes.c

```

00001 /**
00002  * @file Test_Phodiodes.c
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 4/29/2015\n Created: 2/28/2015 7:28:51 PM
00006  * @brief This file is for the testing of the photodiode code.
00007  */
00008
00009 #ifndef F_CPU
00010     /** @brief The current clock speed for the microcontroller. */
00011     #define F_CPU 16000000UL
00012 #endif
00013
00014 #include <util/delay.h>
00015 #include <stdio.h>
00016 #include <stdlib.h>
00017 #include "../Master_Build_Control.h"
00018 #include "../Microcontroller/ADC/CD4051.h"
00019 #include "../Microcontroller/Communication/USART.h"
00020 #include "../Subsystem/Light/CLS15.h"
00021 #include "../Subsystem/Light/XPEBBL.h"
00022
00023 void Test_Phodiodes_Print_Front(void)
00024 {
00025     /* Initialize the temporary variables. */
  
```

```

00026     char Buffer[50] = {0x00};
00027     /* Retrieve the current photodiode ADC value. */
00028     uint16_t Photodiode_ADC = CLS15_Retrieve_Photodiode_ADC_Rear();
00029     /* Print the ADC information. */
00030     sprintf(Buffer,"Front: %u\r\n", Photodiode_ADC);
00031     USART_Send_String(Buffer);
00032 }
00033
00034 void Test_Photodiodes_Print_Left(void)
00035 {
00036     /* Initialize the temporary variables. */
00037     char Buffer[100] = {0x00};
00038     char Buffer_Photodiode_Ratio[FLOATING_POINT_BUFFER_SIZE] = {0x00};
00039     /* Retrieve the current photodiode ADC values and ratio. */
00040     uint16_t Photodiode_ADC_Front = CLS15_Retrieve_Photodiode_ADC_Left_Front
00041 ();
00042     uint16_t Photodiode_ADC_Rear = CLS15_Retrieve_Photodiode_ADC_Left_Rear
00043 ();
00044     float Photodiode_Ratio = CLS15_Retrieve_Photodiode_Ratio_Left();
00045     /* Print the ADC information. */
00046     sprintf(Buffer,"Left Front: %u Rear: %u Ratio: %s\r\n", Photodiode_ADC_Front, Photodiode_ADC_Rear,
00047 USART_ftoa(Photodiode_Ratio, Buffer_Photodiode_Ratio));
00048     USART_Send_String(Buffer);
00049 }
00050
00051 void Test_Photodiodes_Print_Right(void)
00052 {
00053     /* Initialize the temporary variables. */
00054     char Buffer[100] = {0x00};
00055     char Buffer_Photodiode_Ratio[FLOATING_POINT_BUFFER_SIZE] = {0x00};
00056     /* Retrieve the current photodiode ADC values and ratio. */
00057     uint16_t Photodiode_ADC_Front = CLS15_Retrieve_Photodiode_ADC_Right_Front
00058 ();
00059     uint16_t Photodiode_ADC_Rear = CLS15_Retrieve_Photodiode_ADC_Right_Rear
00060 ();
00061     float Photodiode_Ratio = CLS15_Retrieve_Photodiode_Ratio_Right();
00062     /* Print the ADC information. */
00063     sprintf(Buffer,"Right Front: %u Rear: %u Ratio: %s\r\n", Photodiode_ADC_Front, Photodiode_ADC_Rear,
00064 USART_ftoa(Photodiode_Ratio, Buffer_Photodiode_Ratio));
00065     USART_Send_String(Buffer);
00066 }
00067
00068 void Test_Photodiodes_Print_Rear(void)
00069 {
00070     /* Initialize the temporary variables. */
00071     char Buffer[50] = {0x00};
00072     /* Retrieve the current photodiode ADC value. */
00073     int Photodiode_ADC = CLS15_Retrieve_Photodiode_ADC_Rear();
00074     /* Print the ADC information. */
00075     sprintf(Buffer,"Rear: %d\r\n", Photodiode_ADC);
00076     USART_Send_String(Buffer);
00077 }
00078
00079 void Test_Photodiodes_Front(void)
00080 {
00081     /* Initialize the multiplexer. */
00082     CD4051_Initialize();
00083     /* Initialize USART communication. */
00084     USART_Initialize();
00085     while(1)
00086     {
00087         /* Update all of the photodiodes. */
00088         CLS15_Update_Photodiode_Values(8);
00089         /* Print the current value for the front photodiode. */
00090         Test_Photodiodes_Print_Front();
00091         /* Add a line between all of the reads. */
00092         USART_Send_String("\n");
00093         /* Delay between the last check. */
00094         _delay_ms(100);
00095     }
00096 }
00097
00098 void Test_Photodiodes_Left(void)
00099 {
00100     /* Initialize the multiplexer. */
00101     CD4051_Initialize();
00102     /* Initialize USART communication. */
00103     USART_Initialize();
00104     while(1)
00105     {
00106         /* Update all of the photodiodes. */
00107         CLS15_Update_Photodiode_Values(8);
00108         /* Print the current value for the left photodiode. */
00109         Test_Photodiodes_Print_Left();
00110         /* Add a line between all of the reads. */
00111         USART_Send_String("\n");
00112         /* Delay between the last check. */

```

```

00107     _delay_ms(100);
00108     }
00109 }
00110
00111 void Test_Photodiodes_Right(void)
00112 {
00113     /* Initialize the multiplexer. */
00114     CD4051_Initialize();
00115     /* Initialize USART communication. */
00116     USART_Initialize();
00117     while(1)
00118     {
00119         /* Update all of the photodiodes. */
00120         CLS15_Update_Phodiode_Values(8);
00121         /* Print the current value for the right photodiode. */
00122         Test_Photodiodes_Print_Right();
00123         /* Add a line between all of the reads. */
00124         USART_Send_String("\n");
00125         /* Delay between the last check. */
00126         _delay_ms(100);
00127     }
00128 }
00129
00130 void Test_Photodiodes_Rear(void)
00131 {
00132     /* Initialize the multiplexer. */
00133     CD4051_Initialize();
00134     /* Initialize USART communication. */
00135     USART_Initialize();
00136     while(1)
00137     {
00138         /* Update all of the photodiodes. */
00139         CLS15_Update_Phodiode_Values(8);
00140         /* Print the current value for the rear photodiode. */
00141         Test_Photodiodes_Print_Rear();
00142         /* Add a line between all of the reads. */
00143         USART_Send_String("\n");
00144         /* Delay between the last check. */
00145         _delay_ms(100);
00146     }
00147 }
00148
00149 void Test_Photodiodes_All(void)
00150 {
00151     /* Initialize the multiplexer. */
00152     CD4051_Initialize();
00153     /* Initialize USART communication. */
00154     USART_Initialize();
00155     while(1)
00156     {
00157         /* Update all of the photodiodes. */
00158         CLS15_Update_Phodiode_Values(8);
00159         /* Print the current value for the front photodiode. */
00160         Test_Photodiodes_Print_Front();
00161         /* Print the current value for the left photodiode. */
00162         Test_Photodiodes_Print_Left();
00163         /* Print the current value for the right photodiode. */
00164         Test_Photodiodes_Print_Right();
00165         /* Print the current value for the rear photodiode. */
00166         Test_Photodiodes_Print_Rear();
00167         /* Add a line between all of the reads. */
00168         USART_Send_String("\n");
00169         /* Delay between the last check. */
00170         _delay_ms(100);
00171     }
00172 }
00173
00174 void Test_Photodiodes_All_With_LED(void)
00175 {
00176     /* Initialize the multiplexer. */
00177     CD4051_Initialize();
00178     /* Initialize USART communication. */
00179     USART_Initialize();
00180     /* Initialize the LEDs. */
00181     XPEBLL_Initialize();
00182     while(1)
00183     {
00184         /* Change the current LED and all of the photodiodes that are not by the current LED. */
00185         CLS15_Update_Phodiode_Values(
00186             XPEBLL_Change_LED());
00186         /* Print the current value for the front photodiode. */
00187         Test_Photodiodes_Print_Front();
00188         /* Print the current value for the left photodiode. */
00189         Test_Photodiodes_Print_Left();
00190         /* Print the current value for the right photodiode. */
00191         Test_Photodiodes_Print_Right();
00192         /* Print the current value for the rear photodiode. */

```

```

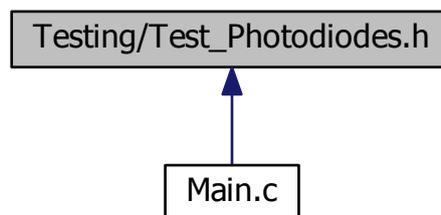
00193     Test_Photodiodes_Print_Rear();
00194     /* Add a line between all of the reads. */
00195     USART_Send_String("\n");
00196     /* Delay between the last check. */
00197     _delay_ms(200);
00198 }
00199 }
00200
00201 /**
00202  * @brief This function is used to test the reading of the photodiodes code.
00203  */
00204 void Test_Photodiodes(void)
00205 {
00206     /* Test the front photodiode. */
00207     #if Photodiode_Test_Mode == 0
00208         Test_Photodiodes_Front();
00209     /* Test the left photodiodes. */
00210     #elif Photodiode_Test_Mode == 1
00211         Test_Photodiodes_Left();
00212     /* Test the right photodiode. */
00213     #elif Photodiode_Test_Mode == 2
00214         Test_Photodiodes_Right();
00215     /* Test the rear photodiode. */
00216     #elif Photodiode_Test_Mode == 3
00217         Test_Photodiodes_Rear();
00218     /* Test the all photodiodes. */
00219     #elif Photodiode_Test_Mode == 4
00220         Test_Photodiodes_All();
00221     /* Test the all photodiodes with LEDs. */
00222     #elif Photodiode_Test_Mode == 5
00223         Test_Photodiodes_All_With_LED();
00224     #endif
00225 }

```

3.100 Testing/Test_Photodiodes.h File Reference

This is the header file for the testing of the photodiode code.

This graph shows which files directly or indirectly include this file:



Functions

- void [Test_Photodiodes](#) (void)

This function is used to test the reading of the photodiodes code.

3.100.1 Detailed Description

This is the header file for the testing of the photodiode code.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 4/11/2015

Created: 2/28/2015 7:29:11 PM

Definition in file [Test_Photodiodes.h](#).**3.100.2 Function Documentation****3.100.2.1 void Test_Photodiodes (void)**

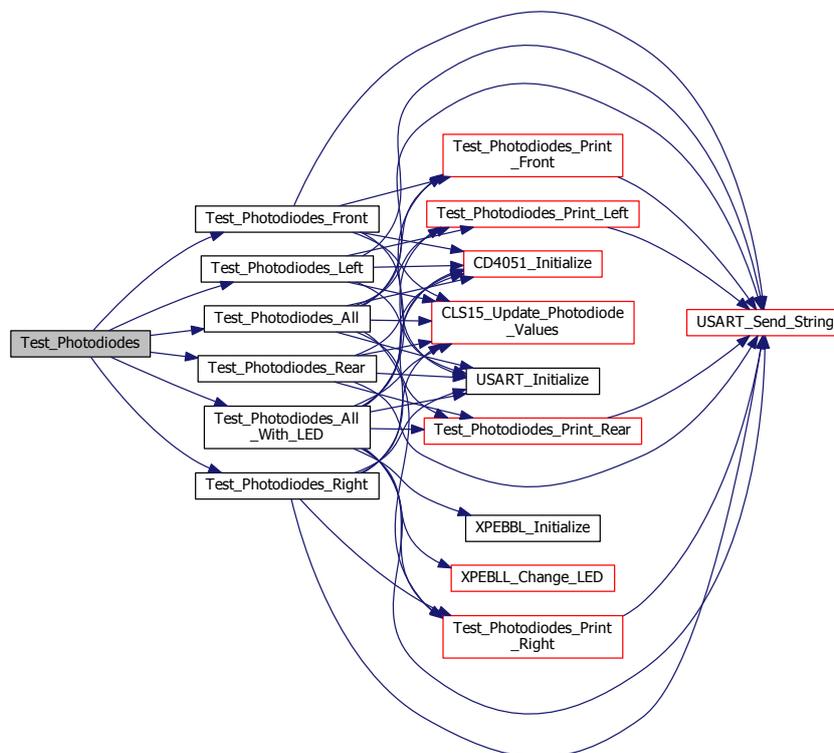
This function is used to test the reading of the photodiodes code.

Definition at line 204 of file [Test_Photodiodes.c](#).

References [Test_Photodiodes_All\(\)](#), [Test_Photodiodes_All_With_LED\(\)](#), [Test_Photodiodes_Front\(\)](#), [Test_Photodiodes_Left\(\)](#), [Test_Photodiodes_Rear\(\)](#), and [Test_Photodiodes_Right\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.101 Test_Photodiodes.h

```

00001 /**
00002  * @file Test_Photodiodes.h
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 4/11/2015\n Created: 2/28/2015 7:29:11 PM
00006  * @brief This is the header file for the testing of the photodiode code.
00007  */
00008
00009 #ifndef TEST_PHOTODIODES_H_
00010 #define TEST_PHOTODIODES_H_
00011
00012 void Test_Photodiodes(void);
00013
00014 #endif /* TEST_PHOTODIODES_H_ */
  
```

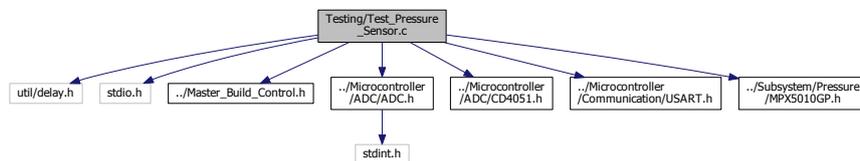
3.102 Testing/Test_Pressure_Sensor.c File Reference

This file is for the testing of the pressure sensor code.

```

#include <util/delay.h>
#include <stdio.h>
#include "../Master_Build_Control.h"
#include "../Microcontroller/ADC/ADC.h"
#include "../Microcontroller/ADC/CD4051.h"
#include "../Microcontroller/Communication/USART.h"
#include "../Subsystem/Pressure/MPX5010GP.h"
  
```

Include dependency graph for Test_Pressure_Sensor.c:



Macros

- #define F_CPU 16000000UL

The current clock speed for the microcontroller.

Functions

- void [Test_Pressure_Sensor](#) (void)

This function is used to test the reading of the pressure sensor.

3.102.1 Detailed Description

This file is for the testing of the pressure sensor code.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 4/29/2015

Created: 2/28/2015 6:41:16 PM

Definition in file [Test_Pressure_Sensor.c](#).

3.102.2 Macro Definition Documentation

3.102.2.1 `#define F_CPU 16000000UL`

The current clock speed for the microcontroller.

Definition at line 11 of file [Test_Pressure_Sensor.c](#).

3.102.3 Function Documentation

3.102.3.1 `void Test_Pressure_Sensor (void)`

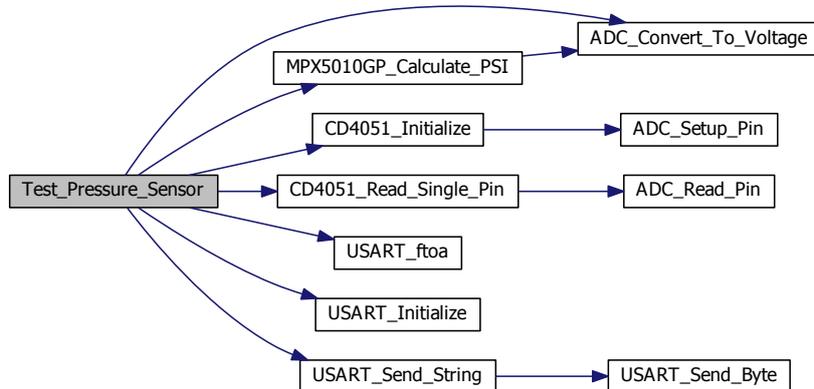
This function is used to test the reading of the pressure sensor.

Definition at line 25 of file [Test_Pressure_Sensor.c](#).

References [ADC_Convert_To_Voltage\(\)](#), [CD4051_Initialize\(\)](#), [CD4051_Read_Single_Pin\(\)](#), [FLOATING_POINT_BUFFER_SIZE](#), [Foot_Water_Per_PSI](#), [MPX5010GP_Calculate_PSI\(\)](#), [Pressure_Sensor_Mux_Port](#), [USART_ftoa\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.103 Test_Pressure_Sensor.c

```

00001 /**
00002  * @file Test_Pressure_Sensor.c
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 4/29/2015\n Created: 2/28/2015 6:41:16 PM
00006  * @brief This file is for the testing of the pressure sensor code.
00007  */
00008
00009 #ifndef F_CPU
00010     /** @brief The current clock speed for the microcontroller. */
00011     #define F_CPU 16000000UL
00012 #endif
00013
00014 #include <util/delay.h>
00015 #include <stdio.h>
00016 #include "../Master_Build_Control.h"
00017 #include "../Microcontroller/ADC/ADC.h"
00018 #include "../Microcontroller/ADC/CD4051.h"
00019 #include "../Microcontroller/Communication/USART.h"
00020 #include "../Subsystem/Pressure/MPX5010GP.h"
00021
00022 /**
00023  * @brief This function is used to test the reading of the pressure sensor.
00024  */
00025 void Test_Pressure_Sensor(void)
00026 {
00027     /* Initialize the multiplexer. */
00028     CD4051_Initialize();
00029     /* Initialize USART communication. */
00030     USART_Initialize();
00031     /* Initialize the temporary variables. */
00032     int ADC_Value = 0;
00033     float ADC_Value_Voltage = 0;
  
```

```

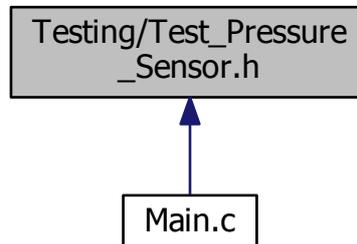
00034     char Buffer_ADC_Value_Voltage[FLOATING_POINT_BUFFER_SIZE] = {0x00};
00035     float PSI_Value = 0;
00036     char Buffer_PSI_Value[FLOATING_POINT_BUFFER_SIZE] = {0x00};
00037     float Temp_Depth = 0;
00038     char Buffer_Temp_Depth[FLOATING_POINT_BUFFER_SIZE] = {0x00};
00039     char Buffer[80] = {0x00};
00040     while(1)
00041     {
00042         /* Read the adc pin value. */
00043         ADC_Value = CD4051_Read_Single_Pin(
Pressure_Sensor_Mux_Port);
00044         /* Convert the adc value to a voltage. */
00045         ADC_Value_Voltage = ADC_Convert_To_Voltage(ADC_Value);
00046         /* Convert the voltage value to PSI. */
00047         PSI_Value = MPX5010GP_Calculate_PSI(ADC_Value);
00048         /* Convert the PSI to a depth. */
00049         Temp_Depth = PSI_Value*Foot_Water_Per_PSI;
00050         /* Print the pressure sensor information. */
00051         sprintf(Buffer,"%d, %s V, %s PSI, %s ft\r\n", ADC_Value, USART_ftoa(ADC_Value_Voltage,
Buffer_ADC_Value_Voltage), USART_ftoa(PSI_Value, Buffer_PSI_Value),
USART_ftoa(Temp_Depth, Buffer_Temp_Depth));
00052         USART_Send_String(Buffer);
00053         /* Delay between the last check. */
00054         _delay_ms(50);
00055     }
00056 }

```

3.104 Testing/Test_Pressure_Sensor.h File Reference

This is the header file for the testing of the pressure sensor code.

This graph shows which files directly or indirectly include this file:



Functions

- void [Test_Pressure_Sensor](#) (void)

This function is used to test the reading of the pressure sensor.

3.104.1 Detailed Description

This is the header file for the testing of the pressure sensor code.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 4/29/2015

Created: 2/28/2015 6:41:35 PM

Definition in file [Test_Pressure_Sensor.h](#).

3.104.2 Function Documentation

3.104.2.1 void Test_Pressure_Sensor (void)

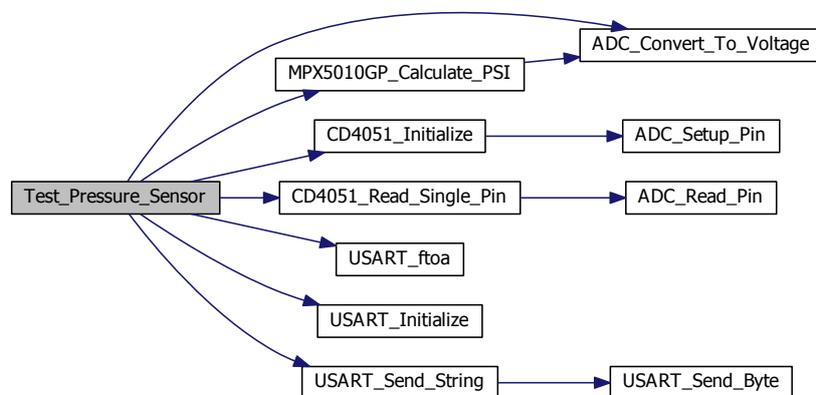
This function is used to test the reading of the pressure sensor.

Definition at line 25 of file [Test_Pressure_Sensor.c](#).

References [ADC_Convert_To_Voltage\(\)](#), [CD4051_Initialize\(\)](#), [CD4051_Read_Single_Pin\(\)](#), [FLOATING_POINT_↔_BUFFER_SIZE](#), [Foot_Water_Per_PSI](#), [MPX5010GP_Calculate_PSI\(\)](#), [Pressure_Sensor_Mux_Port](#), [USART_↔_ftoa\(\)](#), [USART_Initialize\(\)](#), and [USART_Send_String\(\)](#).

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.105 Test_Pressure_Sensor.h

```

00001 /**
00002  * @file Test_Pressure_Sensor.h
00003  * @author Nicholas Sikkema
00004  * @version Revision: 1.0
00005  * @date Last Updated: 4/29/2015\n Created: 2/28/2015 6:41:35 PM
00006  * @brief This is the header file for the testing of the pressure sensor code
00007  */
00008
00009 #ifndef TEST_PRESSURE_SENSOR_H_
00010 #define TEST_PRESSURE_SENSOR_H_
00011
00012 void Test_Pressure_Sensor(void);
00013
00014 #endif /* TEST_PRESSURE_SENSOR_H_ */

```

3.106 Testing/Test_Swarming.c File Reference

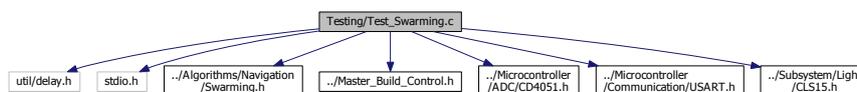
This file is for the testing of the swarming code.

```

#include <util/delay.h>
#include <stdio.h>
#include "../Algorithms/Navigation/Swarming.h"
#include "../Master_Build_Control.h"
#include "../Microcontroller/ADC/CD4051.h"
#include "../Microcontroller/Communication/USART.h"
#include "../Subsystem/Light/CLS15.h"

```

Include dependency graph for Test_Swarming.c:



Macros

- `#define F_CPU 16000000UL`
The current clock speed for the microcontroller.

Functions

- `int Test_Swarming (void)`
This function is used to test the swarming code.

3.106.1 Detailed Description

This file is for the testing of the swarming code.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 4/29/2015

Created: 2/28/2015 5:59:02 PM

Definition in file [Test_Swarming.c](#).

3.106.2 Macro Definition Documentation

3.106.2.1 #define F_CPU 16000000UL

The current clock speed for the microcontroller.

Definition at line 11 of file [Test_Swarming.c](#).

3.106.3 Function Documentation

3.106.3.1 int Test_Swarming (void)

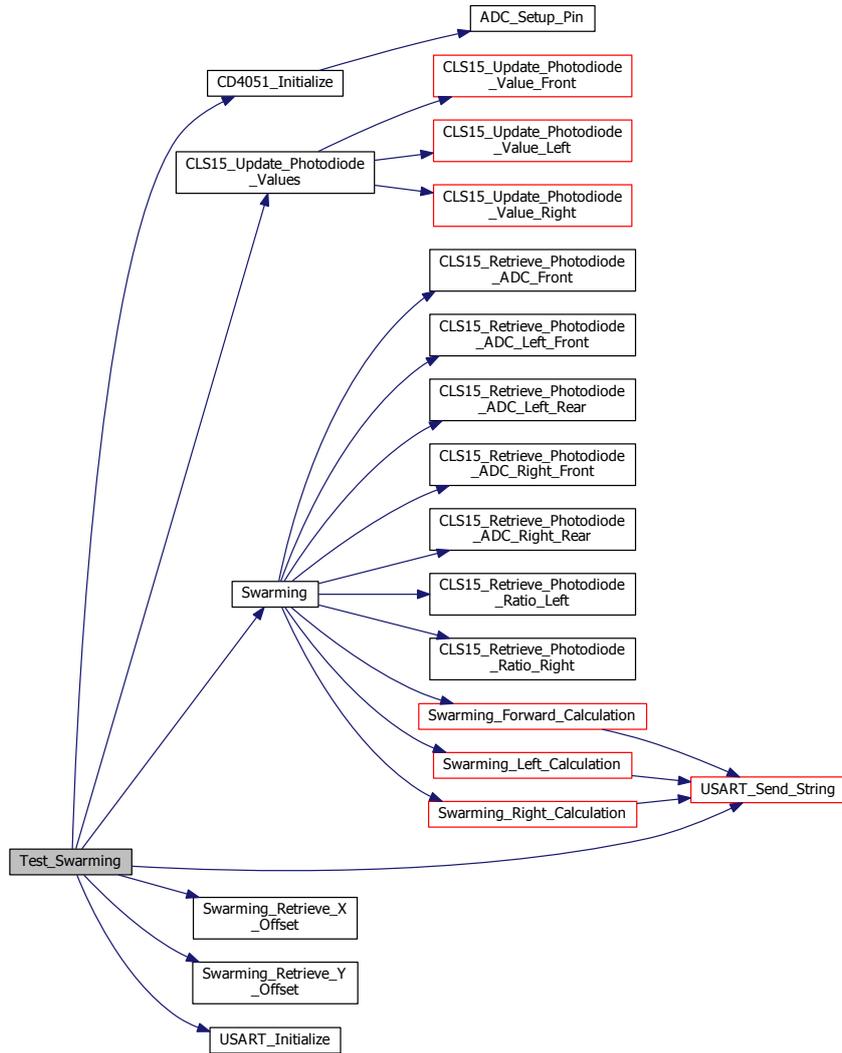
This function is used to test the swarming code.

Definition at line 25 of file [Test_Swarming.c](#).

References [CD4051_Initialize\(\)](#), [CLS15_Update_Phodiode_Values\(\)](#), [Swarming\(\)](#), [Swarming_Retrieve_X_Offset\(\)](#), [Swarming_Retrieve_Y_Offset\(\)](#), [USART_Initialize\(\)](#), [USART_Send_String\(\)](#), X, and Y.

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.107 Test_Swarming.c

```

00001 /**
00002  * @file Test_Swarming.c
00003  * @author Nicholas Sikkema
    
```

```

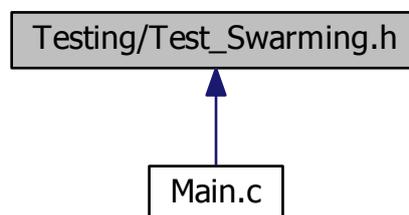
00004 * @version Revision: 1.0
00005 * @date Last Updated: 4/29/2015\n Created: 2/28/2015 5:59:02 PM
00006 * @brief This file is for the testing of the swarming code.
00007 */
00008
00009 #ifndef F_CPU
00010     /** @brief The current clock speed for the microcontroller. */
00011     #define F_CPU 16000000UL
00012 #endif
00013
00014 #include <util/delay.h>
00015 #include <stdio.h>
00016 #include "../Algorithms/Navigation/Swarming.h"
00017 #include "../Master_Build_Control.h"
00018 #include "../Microcontroller/ADC/CD4051.h"
00019 #include "../Microcontroller/Communication/USART.h"
00020 #include "../Subsystem/Light/CLS15.h"
00021
00022 /**
00023  * @brief This function is used to test the swarming code.
00024  */
00025 int Test_Swarming(void)
00026 {
00027     /* Initialize the multiplexer. */
00028     CD4051_Initialize();
00029     /* Initialize USART communication. */
00030     USART_Initialize();
00031     /* Initialize the temporary variables. */
00032     char Buffer[50] = {0x00};
00033     while(1)
00034     {
00035         /* Update the photodiode values. */
00036         CLS15_Update_Phodiode_Values(8);
00037         /* Update the X and Y for the swarm algorithm. */
00038         Swarming();
00039         /* Obtain the X and Y from the swarm algorithm. */
00040         int X = Swarming_Retrieve_X_Offset();
00041         int Y = Swarming_Retrieve_Y_Offset();
00042         /* Print the swarm information. */
00043         sprintf(Buffer,"X: %d, Y: %d\r\n", X, Y);
00044         USART_Send_String(Buffer);
00045         /* Delay between the last check. */
00046         _delay_ms(100);
00047     }
00048 }

```

3.108 Testing/Test_Swarming.h File Reference

This is the header file for the testing of the swarming code.

This graph shows which files directly or indirectly include this file:



Functions

- void [Test_Swarming](#) (void)

This function is used to test the swarming code.

3.108.1 Detailed Description

This is the header file for the testing of the swarming code.

Author

Nicholas Sikkema

Version

Revision: 1.0

Date

Last Updated: 4/29/2015

Created: 2/28/2015 5:59:18 PM

Definition in file [Test_Swarming.h](#).

3.108.2 Function Documentation

3.108.2.1 void Test_Swarming (void)

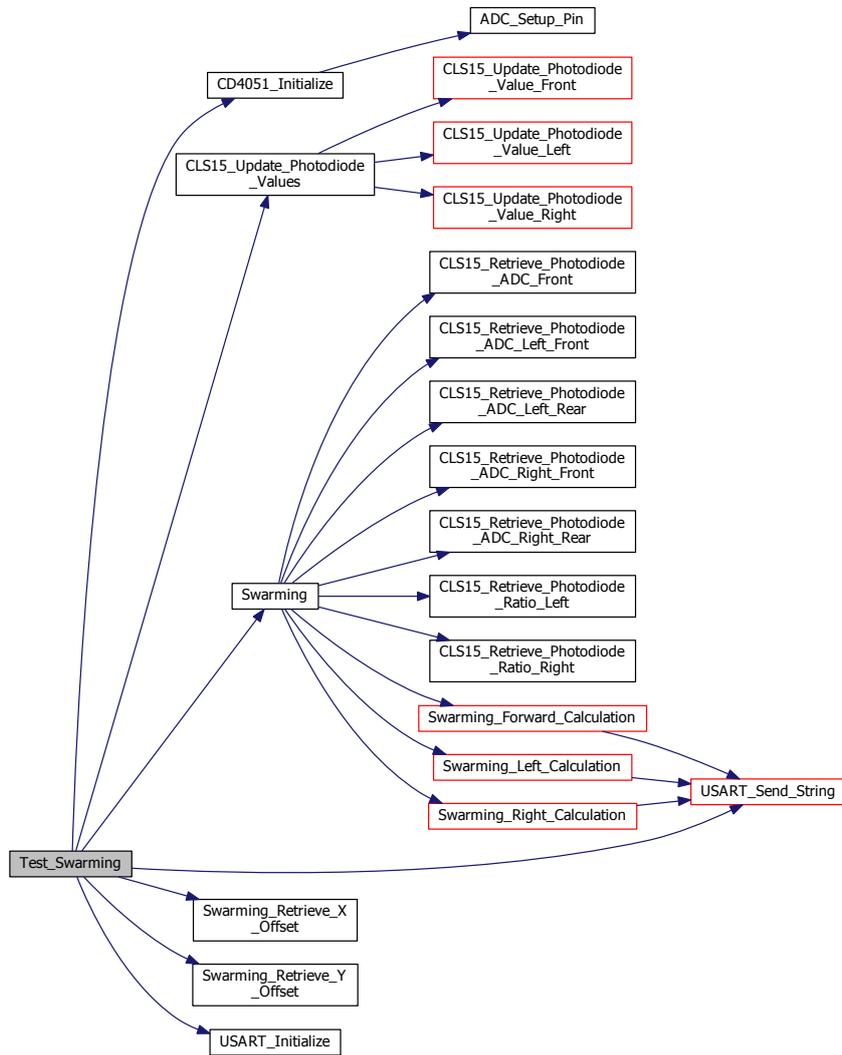
This function is used to test the swarming code.

Definition at line 25 of file [Test_Swarming.c](#).

References [CD4051_Initialize\(\)](#), [CLS15_Update_Phodiode_Values\(\)](#), [Swarming\(\)](#), [Swarming_Retrieve_X_Offset\(\)](#), [Swarming_Retrieve_Y_Offset\(\)](#), [USART_Initialize\(\)](#), [USART_Send_String\(\)](#), X, and Y.

Referenced by [main\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



3.109 Test_Swarming.h

```

00001 /**
00002  * @file Test_Swarming.h
00003  * @author Nicholas Sikkema
    
```

```
00004 * @version Revision: 1.0
00005 * @date Last Updated: 4/29/2015\n Created: 2/28/2015 5:59:18 PM
00006 * @brief This is the header file for the testing of the swarming code.
00007 */
00008
00009 #ifndef TEST_SWARMING_H_
00010 #define TEST_SWARMING_H_
00011
00012 void Test_Swarming(void);
00013
00014 #endif /* TEST_SWARMING_H_ */
```