

STD-BUS TEST SOFTWARE

Reference Manual



Apex Embedded Systems

Test Software Reference Manual

STDBus Isolated Digital Inputs

8-Bit data bus interface

Apex Embedded Systems

116 Owen Road

Monona, WI 53716

Phone 608.256.0767 • Fax 608.256.0765

www.apexembeddedsystems.com

customer.service@apexembeddedsystems.com

Copyright Notice

Copyright © 2015 by Apex Embedded Systems. All rights reserved.

Table of Contents

Welcome	1
Legal Notice	3
C Test Code - DOS	5
Commands	5
Symbol Reference	14
Functions	15
ADS1259_Calibration_Range_Test Function	28
AS_Calibration_Value_Read Function	29
AS_Calibration_Value_Write Function	30
AS_Chip_Select_Route_Override Function	31
AS_Chip_Select_Route_Restore Function	31
AS_Data_Read Function	32
AS_Data_Ready_NoWait Function	33
AS_Data_Ready_Wait Function	33
AS_Opcode_Write Function	34
AS_Rdata_Statistics Function	35
AS_Register_Defaults Function	36
AS_Register_Name_To_Offset Function	36
AS_Registers_Read Function	37
AS_Registers_Write Function	38
AS_Time_Statistics Function	39
Bank_Name_To_Symbol Function	40
Bank_Symbol_To_Name Function	41
Bit_String_Index_By_Name Function	41
Bit_String_Report Function	42
Buffer_Init Function	43
Buffer_Length Function	43
Buffer_Maximum_Int32 Function	44
Buffer_Mean_Int32 Function	44
Buffer_Minimum_Int32 Function	45
Buffer_Peak_To_Peak_Int32 Function	45

Buffer_Save Function	46
Buffer_Save_Binary_Int32 Function	46
Buffer_Standard_Deviation_Int32 Function	47
Buffer_Stuff Function	48
Character_Get Function	48
CMD__AS_Calibration_Gain Function	49
CMD__AS_Calibration_Offset Function	50
CMD__AS_Gancal Function	51
CMD__AS_Ofscal Function	51
CMD__AS_Parameter Function	52
CMD__AS_Rdata Function	54
CMD__AS_Rdatac Function	58
CMD__AS_Reg_Write_By_Name Function	58
CMD__AS_Register_Load Function	60
CMD__AS_Register_Save Function	61
CMD__AS_Reset Function	61
CMD__AS_RReg Function	62
CMD__AS_Sdatac Function	64
CMD__AS_Sleep Function	64
CMD__AS_Start Function	65
CMD__AS_Stop Function	65
CMD__AS_Wakeup Function	66
CMD__AS_WReg Function	66
CMD__FRAM_Dump Function	67
CMD__FRAM_Init Function	68
CMD__FRAM_Load Function	69
CMD__FRAM_RDID Function	70
CMD__FRAM_RDSR Function	70
CMD__FRAM_Save Function	71
CMD__FRAM_WRDI Function	72
CMD__FRAM_WREN Function	72
CMD__FRAM_Write Function	73
CMD__FRAM_WRSR Function	73
CMD__IAI16_AI_All Function	74
CMD__IAI16_AI_Channel Function	75
CMD__IAI16_AI_ID Function	75
CMD__IDI48_DIN_All Function	76
CMD__IDI48_DIN_Channel Function	77

CMD_IDI48_DIN_Group Function	78
CMD_IDI48_DIN_Helper_All_Values_False Function	79
CMD_IDI48_DIN_ID Function	79
CMD_IDI48_DIN_Test Function	80
CMD_IDI48_DIN_Test_FE Function	81
CMD_IDI48_DIN_Test_Interrupt Function	84
CMD_IDI48_DIN_Test_Interrupt_Handler Function	89
CMD_IDI48_DIN_Test_RE Function	90
CMD_IDI48_DIN_Test_Value Function	93
CMDIDO48_DO_All Function	96
CMDIDO48_DO_Channel Function	97
CMDIDO48_DO_Group Function	98
CMDIDO48_DO_ID Function	99
CMDIDO48_DOUT_Test_Alternate Function	100
CMDIDO48_DOUT_Test_One_Hot Function	102
CMDIDO48_IDI48_Loopback Function	103
CMDIDO48_Test Function	109
CMD_Main_AnalogStick_CS Function	110
CMD_Main_Base Function	110
CMD_Main_FRAM_CS Function	111
CMD_Main_I_Count Function	112
CMD_Main_Init_Reg Function	113
CMD_Main_IO_Behavior Function	114
CMD_Main_Irq_Number Function	115
CMD_Main_Mode_Jumpers Function	116
CMD_Main_Trace Function	117
CMD_SPI_Commit Function	118
CMD_SPI_Config_Chip_Select_Behavior Function	118
CMD_SPI_Config_Chip_Select_Route Function	120
CMD_SPI_Config_Clock_Hz Function	120
CMD_SPI_Config_End_Cycle_Delay_Sec Function	121
CMD_SPI_Config_Get Function	122
CMD_SPI_Config_Mode Function	123
CMD_SPI_Config_SDIL_Polarity Function	124
CMD_SPI_Config_SDIO_Wrap Function	125
CMD_SPI_Config_SDO_Polarity Function	126
CMD_SPI_Data Function	126
CMD_SPI_Data_Interpreter Function	128

CMD__SPI_FIFO Function	129
CMD__SPI_ID Function	132
CMD__SPI_Status Function	132
CMD__Trace_File Function	133
CMD__Trace_Start Function	134
CMD__Trace_Stop Function	135
Command_Line_Analog_Stick Function	136
Command_Line_Dump Function	137
Command_Line_FPGA Function	138
Command_Line_FRAM Function	139
Command_Line_Help Function	140
Command_Line_IAI16 Function	140
Command_Line_IDI48 Function	141
Command_LineIDO48 Function	142
Command_Line_IO_Read Function	143
Command_Line_IO_Write Function	145
Command_Line_Loop_Count Function	146
Command_Line_Loop_Delay Function	147
Command_Line_Loop_Space Function	148
Command_Line_Main Function	148
Command_Line_Main_Board_Type_Status Function	149
Command_Line_Prefix Function	150
Command_Line_Prefix_Irq Function	151
Command_Line_Prefix_Loop Function	152
Command_Line_Prefix_Trace Function	153
Command_Line_Register_Transaction Function	154
Command_Line_Set Function	156
Command_Line_SPI Function	157
Command_Line_Wait Function	158
EC_Code_To_Human_Readable Function	158
FPGA_Date_Time_Information Function	159
FRAM__Chip_Select_Route_Override Function	161
FRAM__Chip_Select_Route_Restore Function	162
FRAM__Memory_Read Function	162
FRAM__Memory_Write Function	164
FRAM__Read_ID Function	165
FRAM__Read_Status_Register Function	166
FRAM__Write_Disable Function	167

FRAM__Write_Enable_Latch_Set Function	168
FRAM__Write_Status_Register Function	169
FRAM_File_To_Memory Function	170
FRAM_Memory_To_File Function	171
FRAM_Report Function	172
FRAM_Set Function	173
Help Function	175
Help_Output Function	176
Help_Pause_Helper Function	177
Hex_Dump_Line Function	178
IAI_AI_ID_Get Function	179
IAI16_AI_Channel_Get Function	179
IAI16_AI_Channel_Multiple_Get Function	180
IDI_Bank_Name_To_Symbol Function	181
IDI_Bank_Symbol_To_Name Function	181
IDI_Dataset_Defaults Function	182
IDI_DIN_Channel_Get Function	183
IDI_DIN_Group_Get Function	183
IDI_DIN_ID_Get Function	184
IDI_DIN_IsNotPresent Function	185
IDI_DIN_Pending_Clear Function	185
IDI_Help Function	186
IDI_Initialization_Data_Structure_Load Function	186
IDI_Initialization_Register Function	187
IDI_Main_Loop_Testing Function	188
IDI_Register_Report_CSV Function	189
IDI_Termination Function	190
IDI48_DIN_ID_Port_Scan Function	191
IDO_Bank_Name_To_Symbol Function	192
IDO_Bank_Symbol_To_Name Function	193
IDO_Dataset_Defaults Function	193
IDO_DO_Channel_Get Function	194
IDO_DO_Channel_Set Function	195
IDO_DO_Group_Get Function	195
IDO_DO_Group_Set Function	196
IDO_DOUT_ID_Get Function	196
IDO_DOUT_IsNotPresent Function	197
IDO_Help Function	198

IDO_Initialization_Data_Structure_Load Function	198
IDO_Initialization_Register Function	199
IDO_Register_Report_CSV Function	200
IDO_Termination Function	201
IDO48_IDI48_Loopback_Test Function	202
Initialization Function	202
IO_Direction_IsNotValid Function	204
IO_Get_Symbol_Name Function	204
IO_Read_U16_Address_Fixed Function	205
IO_Read_U16_Address_Increment Function	206
IO_Read_U8 Function	206
IO_Read_U8_Port Function	208
IO_Trace_Dump Function	208
IO_Trace_Initialize Function	211
IO_Trace_Log Function	212
IO_Trace_Log_Dataset Function	212
IO_Trace_Log_Dataset_Begin Function	213
IO_Trace_Log_Dataset_End Function	214
IO_Trace_Log_Function Function	215
IO_Write_U16_Address_Fixed Function	216
IO_Write_U16_Address_Increment Function	216
IO_Write_U8 Function	217
IO_Write_U8_Port Function	218
IOKern_DOS_IRQ_Free Function	219
IOKern_DOS_IRQ_Mask_Get Function	220
IOKern_DOS_IRQ_Mask_Set Function	220
IOKern_DOS_IRQ_Request Function	221
IOKern_DOS_ISR_Install Function	222
IOKern_DOS_ISR_Restore Function	223
IOKern_DOS_Vector_Get Function	224
IOKern_DOS_Vector_Set Function	225
IOKern Interrupt_Helper Function	225
IOKern_IRQ_Invoke_ISR_Test Function	226
IOKern_IRQ_Test Function	226
IOKern_IRQ_Test_Helper Function	227
IOKern_ISR0 Function	228
IOKern_ISR1 Function	228
IOKern_ISR10 Function	229

IOKern_ISR11 Function	229
IOKern_ISR12 Function	230
IOKern_ISR13 Function	230
IOKern_ISR14 Function	231
IOKern_ISR15 Function	231
IOKern_ISR2 Function	232
IOKern_ISR3 Function	232
IOKern_ISR4 Function	233
IOKern_ISR5 Function	233
IOKern_ISR6 Function	234
IOKern_ISR7 Function	234
IOKern_ISR8 Function	235
IOKern_ISR9 Function	235
IOKern_PIC_EOI Function	236
IOKern_Resource_Initialization Function	236
IOKern_Resource_Termination Function	237
IOKern_Task_Handle_Get Function	238
IOKern_Task_ID_Get Function	238
main Function	239
Main_Versalologic Function	243
Print_BytE_List Function	243
Print_Multiple Function	245
Register_Acronym_To_Row Function	245
Register_Report_CSV Function	246
Signal_Handler Function	247
SPI_Calculate_Clock Function	247
SPI_Calculate_End_Cycle_Delay Function	248
SPI_Calculate_Half_Clock Function	249
SPI_Calculate_Half_Clock_Interval_Sec Function	251
SPI_Commit Function	251
SPI_Commit_Get Function	252
SPI_Commit_Is_Inactive Function	253
SPI_Configuration_Chip_Select_Behavior_Get Function	253
SPI_Configuration_Chip_Select_Behavior_Set Function	254
SPI_Configuration_Chip_Select_Route_Get Function	255
SPI_Configuration_Chip_Select_Route_Set Function	256
SPI_Configuration_Get Function	257
SPI_Configuration_Initialize Function	258

SPI_Configuration_Set Function	259
SPI_Data_Read Function	261
SPI_Data_Write Function	262
SPI_Data_Write_Read Function	263
SPI_Data_Write_Read_Helper_Commit Function	270
SPI_Data_Write_Read_Helper_Read Function	271
SPI_Data_Write_Read_Helper_Write Function	273
SPI_FIFO_Read Function	276
SPI_FIFO_Write Function	278
SPI_ID_Get Function	279
SPI_ID_Get_Helper Function	280
SPI_IsNotPresent Function	280
SPI_Report_Configuration_Text Function	281
SPI_Report_Status_Text Function	282
SPI_Status_Read Function	283
SPI_Status_Read_FIFO_Is_Not_Empty Function	284
SPI_Status_Read_FIFO_Status Function	284
SPI_Status_Write Function	286
SPI_Status_Write_FIFO_Is_Full Function	287
SPI_Status_Write_FIFO_Is_Not_Empty Function	288
SPI_Status_Write_FIFO_Status Function	288
StopWatch_Close Function	290
StopWatch_Initialization Function	291
StopWatch_Open Function	291
StopWatch_Process Function	291
StopWatch_Set Function	291
StopWatch_Termination Function	292
StopWatch_TimeStamp_String Function	292
StopWatch_Value Function	292
String_To_Bool Function	293
Termination Function	293
Time_Current_String Function	294
Structs, Records, Enums	295
as_ads1259_registers Structure	297
bank_info Structure	297
bit_string_info Structure	297
board_dataset Structure	298
board_definition Structure	298

buffer_column Enumeration	299
CLM_BTS Enumeration	299
command_line Structure	300
command_line_board Structure	300
din_cfg Structure	301
dout_cfg Structure	301
ec_human_readable Structure	301
idi_bank_info Structure	302
idi_dataset Structure	302
idi_reg_definition Structure	303
ido_bank_info Structure	304
ido_dataset Structure	304
ido_reg_definition Structure	305
io_trace Structure	306
IO_TRACE_ENUM Enumeration	306
io_trace_info Structure	306
IO_TRACE_START_ENUM Enumeration	307
IO_TRACE_STOP_ENUM Enumeration	307
IOKERN_TASK_TYPE Structure	307
irqreturn Enumeration	308
port_list Structure	309
pt_regs Structure	309
reg_definition Structure	310
spi_cfg Structure	310
spi_dataset Structure	311
spi_status Structure	311
timeval Structure	312
BANK_ENUM Enumeration	312
EC_ENUM Enumeration	313
IDI_BANK_ENUM Enumeration	313
IDI_REG_ENUM Enumeration	313
IDO_BANK_ENUM Enumeration	314
IDO_REG_ENUM Enumeration	314
IOKERN_CHAIN_TO_OLD_ENUM Enumeration	314
IOKERN_IRQ_ENUM Enumeration	315
IOKERN_TASK_ID_ENUM Enumeration	316
MODE_JUMPERS_ENUM Enumeration	316
REG_DIR_ENUM Enumeration	317

SPI_CS_B_ENUM Enumeration	317
TEST_STATE_ENUM Enumeration	317
Types	318
BOOL Type	318
int16_t Type	318
int32_t Type	319
int8_t Type	319
IOKERN_DOS_VECTOR_TYPE Type	319
IOKERN_HELP_FP Type	319
IOKERN_ISR_FP Type	320
IOKERN_TASK_FP Type	320
irqreturn_t Type	320
STOPWATCH_TIMEVAL_TYPE Type	321
uint16_t Type	321
uint32_t Type	321
uint8_t Type	321
Variables	322
ADS1259_CLOCK_MHZ Variable	323
ADS1259_CLOCK_TOLERANCE Variable	323
ADS1259_FSC_MAX Variable	324
ADS1259_FSC_MIN Variable	324
ADS1259_OFC_MAX Variable	324
ADS1259_OFC_MIN Variable	324
as_bit_string Variable	325
as_register_name Variable	325
buffer Variable	325
buffer_index Variable	326
cmd_as Variable	326
cmd_fram Variable	327
cmd_iai16 Variable	327
cmd_idi48 Variable	327
cmd_idi48_test Variable	328
cmd_ido48 Variable	328
cmd_ido48_test Variable	328
cmd_loop Variable	329
cmd_prefix Variable	329
cmd_set Variable	329
cmd_spi Variable	330

cmd_top Variable	330
cmd_top__board_type_required Variable	331
cmd_trace Variable	331
ec_unknown Variable	331
global_board_id Variable	332
global_io_trace_buf Variable	332
global_io_trace_enable Variable	332
global_io_trace_name Variable	332
global_io_trace_start_name Variable	333
global_io_trace_stop_name Variable	333
global_irq.Please_install_handler_request Variable	333
global_loop_command Variable	334
global_loop_count Variable	334
global_loop_count_counter Variable	334
global_loop_delay_ms Variable	334
global_loop_space Variable	335
idi_ds Variable	335
ido_ds Variable	335
io kern_isr_table Variable	336
io kern_task Variable	336
lpm_lx800_port_list Variable	336
svn_revision_string Variable	337
Macros	337
__STOPWATCH_H__ Macro	340
AS_ADS1259_REGISTER_QTY Macro	340
BANK_EXTRACT_DEFINITION Macro	340
BANK_EXTRACT_ENUM Macro	341
BANK_INFO_DEFINITION Macro	341
BANK_NULL_DEFINITION Macro	341
BUFFER_COLUMN Macro	342
BUFFER_LENGTH Macro	342
CLOCK_PERIOD_SEC Macro	342
CMD_MAIN_EXTRACT_BOARD_TYPE_REQUIRED Macro	342
CMD_MAIN_EXTRACT_COMMANDS Macro	343
CMD_MAIN_LIST Macro	343
DIN_TEST_DEBUG_PRINT Macro	344
EC_EXTRACT_ENUM Macro	344
EC_EXTRACT_HUMAN_READABLE Macro	344

EC_HUMAN_READABLE_TERMINATE Macro	345
ERROR_CODES Macro	345
false Macro	346
FRAM_BLOCK_SIZE Macro	346
IAI_REGISTER_SET_DEFINITION Macro	346
IDI_BANK_INFO_DEFINITION Macro	348
IDI_BOARD_INIT_FILE_NAME Macro	348
IDI_DIN_GROUP_QTY Macro	349
IDI_DIN_GROUP_SIZE Macro	349
IDI_DIN_QTY Macro	349
IDI_DIN_SHIFT_RIGHT Macro	349
IDI_IO_DIRECTION_TEST Macro	350
IDI_REGISTER_SET_DEFINITION Macro	350
IDO_BANK_INFO_DEFINITION Macro	352
IDO_BOARD_INIT_FILE_NAME Macro	353
IDO_DO_GROUP_QTY Macro	353
IDO_DO_GROUP_SIZE Macro	353
IDO_DO_QTY Macro	353
IDO_DO_SHIFT_RIGHT Macro	354
IDO_REGISTER_SET_DEFINITION Macro	354
INT32_MAX Macro	355
INT32_MIN Macro	355
INTERRUPT Macro	356
IO_TRACE_DEFINITION Macro	356
IO_TRACE_DUMP_FILE_NAME Macro	356
IO_TRACE_EXTRACT_ENUM Macro	357
IO_TRACE_EXTRACT_NAME Macro	357
IO_TRACE_FILE_NAME_SIZE Macro	357
IO_TRACE_SIZE Macro	357
IO_TRACE_START_DEFINITION Macro	358
IO_TRACE_START_STOP_EXTRACT_ENUM Macro	358
IO_TRACE_START_STOP_EXTRACT_NAME Macro	358
IO_TRACE_START_STOP_EXTRACT_NAME_HELP_LIST Macro	359
IO_TRACE_START_STOP_EXTRACT_SWITCH_STATEMENT Macro	359
IO_TRACE_STOP_DEFINITION Macro	359
IO_TRACE_USE_AS_LOGGING Macro	360
IO_TRACE_USE_FRAM_LOGGING Macro	360
IO_TRACE_USE_SPI_LOGGING Macro	360

IOKERN_DOS_INT_GET Macro	360
IOKERN_DOS_INT_SET Macro	361
IOKERN_DOS_NSEOI Macro	361
IOKERN_DOS_PIC1_BASE_ADDRESS Macro	361
IOKERN_DOS_PIC1_CMD Macro	362
IOKERN_DOS_PIC1_IMR Macro	362
IOKERN_DOS_PIC2_BASE_ADDRESS Macro	362
IOKERN_DOS_PIC2_CMD Macro	362
IOKERN_DOS_PIC2_IMR Macro	363
IOKERN_IRQ_CHAIN_SELECT Macro	363
IOKERN_IRQ_ENABLE Macro	364
IOKERN_IRQ_END Macro	364
IOKERN_IRQ_START Macro	364
IOKERN_LOCAL_IRQ_RESTORE Macro	364
IOKERN_LOCAL_IRQ_SAVE Macro	365
IOKERN_TASK_QTY Macro	365
MESSAGE_SIZE Macro	366
REG_EXTRACT_DEFINITION Macro	366
REG_EXTRACT_ENUM Macro	366
REG_LOCATION_CHANNEL_GET Macro	367
REG_LOCATION_LOGICAL_GET Macro	367
REG_LOCATION_SET Macro	367
REGS_INIT_QTY Macro	367
SPI_BLOCK_SIZE Macro	368
SPI_COMMIT_BIT_POSITION Macro	368
SPI_REGISTER_SET_DEFINITION Macro	368
STOPWATCH_ERROR Macro	369
STOPWATCH_HANDLE_TYPE Macro	369
STOPWATCH_INACTIVE Macro	370
STOPWATCH_PAUSE Macro	370
STOPWATCH_QUANTITY Macro	370
STOPWATCH_RESET Macro	370
STOPWATCH_START Macro	371
STOPWATCH_STATE_TYPE Macro	371
strcmpl Macro	371
SVN_REV Macro	372
TARGET_CPU_INTEL_386 Macro	372
true Macro	372

UINT32_MAX Macro	372
Files	373
idi.c	373
stopwatch.h	580

Index

a

STD-Bus Test Software Reference Manual

1 Welcome

Dear Valued Customer:

Thank you for your interest in our products and services.

Apex Embedded Systems "Continuous improvement" policy utilizes customer feedback to improve existing products and create new product offerings based on the needs of our customers.

Continued Success,

Apex Embedded Systems

2 Legal Notice

Apex Embedded Systems' sole obligation for products that prove to be defective within 1 year from date of purchase will be for replacement or refund. Apex Embedded Systems gives no warranty, either expressed or implied, and specifically disclaims all other warranties, including warranties for merchantability and fitness. In no event shall Apex Embedded Systems' liability exceed the buyer's purchase price, nor shall Apex Embedded Systems be liable for any indirect or consequential damages.

This warranty does not apply to products which have been subject to misuse (including static discharge), neglect, accident or modification, or which have been soldered or altered during assembly and are not capable of being tested.

DO NOT USE PRODUCTS SOLD BY APEX EMBEDDED SYSTEMS AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS!

Products sold by Apex Embedded Systems are not authorized for use as critical components in life support devices or systems. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

3 C Test Code - DOS

Symbol Reference

Name	Description
Functions (see page 15)	The following table lists functions in this documentation.
Structs, Records, Enums (see page 295)	The following table lists structs, records, enums in this documentation.
Types (see page 318)	The following table lists types in this documentation.
Variables (see page 322)	The following table lists variables in this documentation.
Macros (see page 337)	The following table lists macros in this documentation.
Files (see page 373)	The following table lists files in this documentation.

3.1 Commands

NOTE:

1. This software will merge into a more general structure to be used by all boards.
2. Refer to latest software for latest commands. This section may not be totally up to date.

This software is set up to test the following boards:

- STDBus Isolated Digital Input Board (IDI48)
- STDBus Isolated Digital Output Board (IDO48)

The current application runs in DOS. Later it will be retargetted to MQX running on the K64F CPU board (ARM32).

It is possible to write shell or batch scripts to build more complex testing scenarios which is supported via the "sh" command in MQX command line processor.

Notes

1. the '[' and ']' are used to indicate an optional parameter.
2. the '<' and '>' indicate a required parameter
3. <type> to be replaced by "idi" or "ido" or "IDI" or "IDO" as the board type (without the quotes).
4. Software intended to drive only one board of each type.
5. <bool> can be any of the following "0", "1", "t", "T", "f", "F". The string can be more than one character, only the first character is tested.

Command prefix

Command prefixes allow for a variety of loops, tracing and so on to only be enabled for a particular command issued.

b [prefix - see table below]remaining command.....

Command Prefix	Description
[loop [count <value>] [delay <milliseconds>] [space]]	<p>All commands except 'help' can have a "loop" prefix such that the command is issued over and over until a key is pressed.</p> <p>Example:</p> <p>Loop as fast as you can or until a key is pressed at terminal: b loop di all binary</p> <p>Loop with a 100mS delay or until a key is pressed at terminal: b loop delay 100 di all binary</p> <p>Loop as fast as you can 10 times or until a key is pressed at terminal: b loop count 10 di all binary</p> <p>Loop with a 100mS delay 10 times or until a key is pressed at terminal: b loop delay 100 count 10 di all binary or b loop count 10 delay 100 di all binary</p> <p>Loop by cycling through same command when SPACEBAR is pressed: b loop space di all binary</p>
[trace]	<p>All commands except 'help' can have a "trace" prefix, this means that it is possible to log I/O transactions.</p> <p>Example:</p> <p>Log a list of I/O transactions when retrieving the SPI interface ID value: b trace idi spi id</p>

<p>[irq]</p> <p>All IDI48 related commands except 'help' can pick up the predefined irq provided the "irq" prefix within the command is issued.</p> <p>Example:</p> <p>Example #1: "idi irq loop din all binary". In this case we allow an interrupt to be used by the "din all" function. Example #2: "idi irq din all binary". In this case we allow an interrupt to occur during the short run of the "din all" function.</p>

Command List

Revised July 15, 2015 - combining IDO48 and IDI48 commands.

Revised August 15, 2015 - Added trace functionality

Command	Description
::::::: HELP :::::::	::::::: HELP :::::::
b [help] [<file name>]	Outputs human readable help to the terminal. If a file name is included in the command the help will be dumped directly to the file, otherwise it is dumped to the default terminal twenty lines at a time.
b <type> dump [<file name>]	Outputs a textual list of register information in CSV format. If file name is not specified it is dumped to the terminal.
::::::: SET :::::::	::::::: SET ::::::: Note: These are application layer environment variables that are persistent.
b <type> <register acronym> [<value>]	Reads or writes a register to/from the board specified. If <value> is not present, then a read is assumed. The ability to read or write is defined by the register set read/write column. The register acronym along with read/write access can be found by referring to IDO_REGISTER_SET_DEFINITION (see page 354) and IDO_REGISTER_SET_DEFINITION (see page 354) for further details or the register set summary.
b <type> set	Reports most of the parameters that are stored in IDO_INIT.BIN or IDO_INIT.BIN file.
b <type> set	Reports most of the parameters that are stored in the IDO_INIT.BIN file.
b <type> set base	Reports the base address. This is stored in a binary file which will NOT be compatible between 16-bit and 32-bit. However, it is easily reproduced.
b <type> set base <address>	Sets the base address from which we will use for all additional commands. This is stored in a binary file which will NOT be compatible between 16-bit and 32-bit due to padding. However, it is easily reproduced.
b <type> set io	The "io" commands are intended for debugging purposes including situations where there is no hardware. reports I/O simulation and reporting modes.
b <type> set io simulate	reports the state of I/O simulation. If "false", then we write to hardware, otherwise it prints the transaction to the terminal.
b <type> set io simulate <true/false>	sets the I/O simulation mode.

b <type> set io report	reports the state of I/O reporting. If "false", then I/O is not reported, otherwise any I/O transaction is reported to the terminal.
b <type> set io report <true/false>	sets the I/O reporting.
b idi set irq [<irq number>]	Writes or reads the IRQ number default for the application to use.
b idi set iqty [<irq count quantity>]	Writes or reads the maximum IRQ count before software exists. Currently, not used by any of the application.
b <type> set framcs [<0/1>]	Write or read the FRAM over-ride SPI chip select route (either CS0 or CS1). The chip select routing is applied temporarily over SPI chip select routing. After the FRAM transaction is complete the SPI chip select route is restored.
b <type> set aics [<0/1>]	Write or read the Analog Stick over-ride SPI chip select route (either CS0 or CS1). The chip select routing is applied temporarily over SPI chip select routing. After the Analog Stick transaction is complete the SPI chip select route is restored.
b <type> set mode [<legacy/0/1/2/3>]	Sets the mode jumpers as stuffed onto the board. The word "legacy" can also be used to represent mode "0".
b <type> set as [<params>]	used to build the register or bit level parameters for the ADS1259 that will be used to initialize the hardware. Refer to the as param command for further details.
b <type> trace	Returns all trace parameters. The purpose of the trace mechanism is to provide a high speed recording of the I/O bottom transactions (primarily). The depth is limited by
b <type> trace start [<disable/on/error/spi/fram/as>]	Set or return the trace start behavior. Possible entries are: disable ==> no logging. on ==> begins logging immediately error ==> begin logging on any error spi ==> begin logging on any SPI transactions fram ==> begin logging on any FRAM transactions as ==> begin logging on any Analog Stick transactions
b <type> trace stop [<none/error/ctrlc>]	Set or return the trace stop behavior. Possible entries are: none ==> never stop logging until end of executable. error ==> stop logging on an error ctrlc ==> stop logging on a break CTRL-C
b <type> trace file [<file_name>]	Set or return the file name in which the trace information will be dumped too. File is overwritten each time the executable is run.
::::::: IDI48 :::::::	
b [idi] di id	reports the digital input module ID as a 16-bit value. This can be considered the board revision/ID as well.
b [idi] di	reports all digital input values
b [idi] di all [<binary>] [<hex/group>]	reports all digital input values. If 'binary' specified then only binary representation is output. If 'hex' or 'group' is specified then only the hexadecimal byte values are output. If both or neither parameters are specified, then both formats are output.
b [idi] di chan <n>	Reports a digital input value at a given channel, where 0 <= n <= 47
b [idi] di group	Reports all digital inputs at all groups

b [idi] di group all	Reports all digital inputs at all groups
b [idi] di group <n>	Reports a byte of digital inputs at the given group, where 0 <= n <= 5
b [idi] di test value [<--append>] [<result file>]	As each input is toggled, the software logs until all inputs have been successfully toggled. Once all inputs are toggled, a "PASS" string is indicated. Pressing any key will stop the execution of the command followed by a "FAIL" string.
b [idi] di test re	This test is checking specifically for rising-edge detection (LED perspective). As each input is toggled, the software logs until all inputs have been successfully toggled. Once all inputs are toggled, a "PASS" string is indicated. Pressing any key will stop the execution of the command followed by a "FAIL" string.
b [idi] di test fe	This test is checking specifically for falling-edge detection (LED perspective). As each input is toggled, the software logs until all inputs have been successfully toggled. Once all inputs are toggled, a "PASS" string is indicated. Pressing any key will stop the execution of the command followed by a "FAIL" string.
b [irq] [idi] di test interrupt [<irq #>]	This test is checking specifically for rising-edge detection (LED perspective). This time interrupts are activated and used for detection. As each input is toggled, the software logs until all inputs have been successfully toggled. Once all inputs are toggled, a "PASS" string is indicated. Pressing any key will stop the execution of the command followed by a "FAIL" string. Optional IRQ number can be given, otherwise it reverts to the IRQ number specified in the "idi set irq"
..... IDO48 IDO48
b [ido] do chan <n> [<bool>]	Read or write to the digital output channel, where 0 <= n <= 47.. If the boolean value is missing it is assumed to be a read, otherwise a write will occur.
b [ido] do group [<n> [<value> [<value> [<value> [<value> [<value> [<value>]	Read or write to the digital output group (8-bit data). If no values are present then it is assumed to be a read, otherwise a byte write operation will occur starting at group position n up to the last possible group number. Note that 0 <= n <= 5. Example: b do group 4 0xAA 0x55 In this case it is writing to register DOG4 and DOG5.
b [ido] do all [<binary> [<hex/group>]	Reports all digital output values in the format requested. If 'binary' specified then only binary representation is output. If 'hex' or 'group' is specified then only the hexadecimal byte values are output. If both or neither parameters are specified, then both formats are output.
b [ido] do test onehot [<on_time>] [<off_time>]	Toggles each individual output one at a time. This test can be exited at any time by pressing any key. This can be easily observable using an LED panel.
b [ido] do test alternate <pattern> time [<time_normal> [<time_inverted>]	Outputs pattern on all groups (or ports). The pattern will be inverted.
b [ido] do test loopback [<--append>] [<result file>]	Assumes IDI48 and interconnection cables have been installed. It assumes that the IDI48 base address has been properly set. Output is directly to terminal or to file.
..... SPI SPI Communications
b <type> spi id	reports the spi wishbone component ID
b <type> spi cfg	reports the spi configuration as it exists within the hardware

b <type> spi clk	reports the spi clock frequency in hertz
b <type> spi clk <freq hz>	set the spi clock frequency in hertz
b <type> spi ecd	reports the spi end of cycle delay in seconds
b <type> spi ecd <time in seconds>	set the spi end of cycle delay in seconds
b <type> spi mode	reports the spi mode (i.e. clock phase and polarity combination)
b <type> spi mode <0/1/2/3>	set the spi mode, where the value is either 0,1,2, or 3.
b <type> spi sdi	reports the spi sdi polarity. SDI is the serial data input.
b <type> spi sdi <true/1/false/0>	sets the spi sdi polarity
b <type> spi sdo	reports the spi sdo polarity. SDO is the serial data output.
b <type> spi sdo <true/1/false/0>	sets the spi sdo polarity.
b <type> spi wrap	reports the spi wrap. If wrap='1' then the SDO is routed to SDI internally (i.e. loopback).
b <type> spi wrap <true/1/false/0>	sets the spi wrap, where false/0 => no wrap, true/1 => wrap.
b <type> spi csr	reports the spi chip select routing. If '1', then chip select routed to physical CS1, otherwise CS0.
b <type> spi csr <true/1/false/0>	set the spi chip select routing. If true or '1', then chip select is routed to physical CS1, otherwise CS0.
b <type> spi csb	reports the spi chip select behavior
b <type> spi csb <0/1/2/3>	sets the spi chip select behavior
b <type> spi status	reports the spi read fifo status
b <type> spi status rx	reports the spi read fifo status
b <type> spi status tx	reports the spi write fifo status
b <type> spi status [tx] [rx]	reports either tx fifo status or rx fifo status or both.
b <type> spi data <character/number> [<character/number>]	write/read one or more bytes. If the fifo becomes full prior to writing all the values, the remaining values will be ignored. Examples: write a single data byte and get a single byte in return. idi spi data 0x55 write 5 characters and receive five characters idi spi data tx h e l l o
b <type> spi fifo	reads out the remaining data sitting in the spi read fifo (until empty) and produces a report.
b <type> spi fifo rx	reads out the remaining data sitting in the spi read fifo (until empty) and produces a report.
b <type> spi fifo rx [<number>]	reads out the remaining 'number' of bytes from the receive/read fifo (until empty) and produces a report.

b <type> spi fifo tx <character/number> [<character/number>]	writes data to the transmit/write fifo. If the fifo becomes full prior to writing all the values, the remaining values will be ignored. Examples: write a single data byte: idi spi data tx 0x55 write 5 characters to the fifo idi spi data tx h e l l o The data can be single characters, decimal or hex numbers. Examples: a, 0x55, 125,. The data can also be a double quote encased string. Example: "this is a sample string".
b <type> spi fifo commit [<1/0>]	Depending on chip select behavior this will either set/clear the chip select or tell the SPI module to begin transmitting any data sitting in the write/transmit FIFO.
::::::: FRAM :::::::	::::::: FRAM using SPI ::::::: Notes: <ol style="list-style-type: none"> 1. Resistors must be installed correctly for FRAM to work correctly. 2. The "framcs" must be set correctly to point to the correct chip select channel.
b <type> fram dump <address> [<length>]	reports the data from the starting address of the FRAM for a given length or number of bytes. The report is typical of any memory dump where it posts for each line a starting address, up to 16 hexadecimal values followed by ASCII representation. If length is not specified, then up to 16-bytes (or until end of the FRAM) is reported. This report does not wrap around the end of the FRAM, just reports up to the end.
b <type> fram save <address> <length> <binary file name>	Saves the portion of FRAM to a given file name. There is no limit on the length as the range can be from 0 (nothing written) to FRAM_DENSITY_BYTES (=8192 bytes).
b <type> fram load <address> <binary file name>	Reads data from the binary file and loads it into the FRAM starting at the address location. If the length is set to 0, then it will read all of the file up to the end of the FRAM. If length is specified, then it is possible to only load a portion of a binary file.
b <type> fram init [<character/number> [<character/number>]]	Initializes the FRAM with a given pattern. If no pattern is specified, it will be filled with all zeros. The pattern can be any number of characters. Only the first 16 values will be used, any additional values will be ignored. This allows one to create a unique pattern especially for implementing a memory manager to manage the FRAM memory. The data can be single characters, decimal or hex numbers. Examples: a, 0x55, 125,. The data can also be a double quote encased string. Example: "this is a sample string".
b <type> fram write <address> [<character/number> [<character/number>]]	Write data to the FRAM give a starting address. The data can be single characters, decimal or hex numbers. Examples: a, 0x55, 125,. The data can also be a double quote encased string. Example: "this is a sample string".
b <type> fram wren	Sets the write enable latch of the FRAM. Note: WREN = WRite ENable.
b <type> fram wrdi	Resets the write enable latch of the FRAM. Note: WRDI = WRite DIable.

b <type> fram rdsr	<p>Read and report FRAM status register.</p> <p>Status register format:</p> <table border="0"> <tr><td>BIT DESCRIPTION</td></tr> <tr><td>7 WPEN - status register write protect.</td></tr> <tr><td>6..4 not used</td></tr> <tr><td>3 BP1 - block protect</td></tr> <tr><td>2 BP0 - block protect</td></tr> <tr><td>1 WEL - write latch enable</td></tr> <tr><td>0 0 - fixed as zero</td></tr> </table> <p>Please refer to the datasheet.</p>	BIT DESCRIPTION	7 WPEN - status register write protect.	6..4 not used	3 BP1 - block protect	2 BP0 - block protect	1 WEL - write latch enable	0 0 - fixed as zero
BIT DESCRIPTION								
7 WPEN - status register write protect.								
6..4 not used								
3 BP1 - block protect								
2 BP0 - block protect								
1 WEL - write latch enable								
0 0 - fixed as zero								
b <type> fram wrsr <status byte>	<p>Write to the FRAM status register.</p> <p>Status register format:</p> <table border="0"> <tr><td>BIT DESCRIPTION</td></tr> <tr><td>7 WPEN - status register write protect.</td></tr> <tr><td>6..4 not used</td></tr> <tr><td>3 BP1 - block protect</td></tr> <tr><td>2 BP0 - block protect</td></tr> <tr><td>1 WEL - write latch enable</td></tr> <tr><td>0 0 - fixed as zero</td></tr> </table> <p>Please refer to the datasheet.</p>	BIT DESCRIPTION	7 WPEN - status register write protect.	6..4 not used	3 BP1 - block protect	2 BP0 - block protect	1 WEL - write latch enable	0 0 - fixed as zero
BIT DESCRIPTION								
7 WPEN - status register write protect.								
6..4 not used								
3 BP1 - block protect								
2 BP0 - block protect								
1 WEL - write latch enable								
0 0 - fixed as zero								
b <type> fram rdid	<p>report the FRAM ID register. The FRAM we are using contains such register, most others do not. The output is a 32-bit hex value of the format: <manufacture_id><continuation code><product ID MSB><Product ID LSB>.</p> <p><manufacture_id> = 0x04 (Fujitsu). <continuation code> = 0x7F (fixed constant). <product ID MSB> = 0x03 (indicating 64kbit density) <product ID LSB> = 0x02 (proprietary information)</p> <p>Please refer to the datasheet.</p>							
::::::: ANALOG STICK :::::::	<p>::::::: ANALOG STICK using SPI :::::::</p> <p>Notes:</p> <ol style="list-style-type: none"> 1. Resistors must be installed correctly for Analog Stick to work correctly. 2. The "aics" must be set correctly to point to the correct chip select channel. 							
b <type> as wakeup	Exit Sleep Mode. Sends a single byte (0x03) instruction to the ADS1259 to exit low-power SLEEP mode.							
b <type> as sleep	Enter Sleep Mode. Sends a single byte (0x05) instruction to the ADS1259 to enter low-power SLEEP mode							
b <type> as reset	Reset Registers to Default Values. Sends a single byte (0x07) instruction to the ADS1259 to reset the digital filter cycle and returns all registers to their default values.							

b <type> as start	Start Conversions. Sends a single byte (0x09) instruction to the ADS1259 start conversion. If CONFIG2.PULSE='1' then a single conversion is performed. If CONFIG2.PULSE='0' (default), then conversions continue until the STOP command is received.
b <type> as stop	Stop Conversions. Sends a single byte (0x0B) instruction to the ADS1259 to stop conversions. If a conversion is in progress it will run to completion and then stop.
b <type> as rdatac	Read Data Continuous. Sends a single byte (0x10) instruction to the ADS1259 to enable reading data in continuous mode. Disable this mode by sending the SDATAC command and before sending other commands.
b <type> as sdatac	Stop Read Data Continuous. Sends a single byte (0x11) instruction to the ADS1259 to cancel or stop reading data continuously.
b <type> as rdata [<count>] [<text <file name>] [--v] [--binary <file name>] [--e]	Read Data. Sends a single byte (0x13) instruction to the ADS1259 to read conversion result (in Stop Read Data Continuous mode). --v Optionally add verbosity (i.e. more output to terminal). --binary <file name> Optionally write to a binary file. --text <file name> Optionally write to a text file --e Optionally pull register configuration from environment
b <type> as rreg <start_address> <register_count> [<hex/symbol>]	Read from Register(s). Sends opcode instruction to the ADS1259 and receives ADS1259 register data based on <start_address> and <register_count>. Symbol names are the following: config0 config1 config2 ofc0 ofc1 ofc2 fsc0 fsc1 fsc2
b <type> as rreg all [<hex/symbol>]	
b <type> as wreg <start_address> <value> [<value>]	Write to Register(s). Sends one or more bytes to the ADS1259 register(s).
b <type> as wrn <reg_name> <value> [<reg_name> <value>]	Write to one or more ADS1259 registers. There must be a pair of values for each register (i.e. register name and value). Register names: config0 config1 config2 ofc0 ofc1 ofc2 fsc0 fsc1 fsc2
b <type> as fsc [<uint32_t (see page 321) value>]	Full scale calibration value, read/write access.
b <type> as ofc [<uint32_t (see page 321) value>]	Offset calibration, read/write access.
b <type> as rsave <file name>	Save all ADS1259 registers to a file.

b <type> as rload [<file name>]	Load all ADS1259 registers from file. If file is incorrect, then registers are not disturbed. If file name not specified, then it will load from environment.
b <type> as param <name> [<value>]	Used to read/write parameter from/to the ADS1259. It is possible to read/write the calibration values, bit-strings or registers. Below is a list of possible parameters that can be read/written. <pre>ofc [<int32_t (□ see page 319)>] offset calibration, range: -8,388,607 to 8,388,607 fsc [<int32_t (□ see page 319)>] full scale calibration, range: 0 (x0) to 8388608 (x2) config0 [<uint8_t (□ see page 321)>] config1 [<uint8_t (□ see page 321)>] config2 [<uint8_t (□ see page 321)>] ofc0 [<uint8_t (□ see page 321)>] ofc1 [<uint8_t (□ see page 321)>] ofc2 [<uint8_t (□ see page 321)>] fsc0 [<uint8_t (□ see page 321)>] fsc1 [<uint8_t (□ see page 321)>] fsc2 [<uint8_t (□ see page 321)>] spi [<0-1>] rbias [<0-1>] id delay [<0-7>] extref [<0-1>] sinc2 [<0-1>] chksum [<0-1>] flag [<0-1>] dr [<0-7>] pulse [<0-1>] syncout [<0-1>] extclk [<0-1>] drdy_b</pre>
::::::: MISC COMMANDS	::::::: MISC COMMANDS - NONE BOARD SPECIFIC :::::::
b o <address> <data>	Unrestricted I/O write. The data is a byte.
b i <address>	Unrestricted I/O read. Will return a byte value.
b [<type>] wait	Simply waits until any key is pressed.

3.2 Symbol Reference

3.2.1 Functions

The following table lists functions in this documentation.

Functions

	Name	Description
≡	ADS1259_Calibration_Range_Test (see page 28)	brief
≡	AS_Calibration_Value_Read (see page 29)	brief
≡	AS_Calibration_Value_Write (see page 30)	brief
≡	AS_Chip_Select_Route_Override (see page 31)	brief Analog Stick SPI chip select over-ride mechanism. Temporarily used to set the chip select channel to what we need based on "set framcs".
≡	AS_Chip_Select_Route_Restore (see page 31)	brief Analog Stick SPI chip select restore mechanism. Returns the chip select to original SPI configuration settings.
≡	AS_Data_Read (see page 32)	brief Rdata command
≡	AS_Data_Ready_NoWait (see page 33)	brief
≡	AS_Data_Ready_Wait (see page 33)	brief
≡	AS_Opcode_Write (see page 34)	brief
≡	AS_Rdata_Statistics (see page 35)	brief
≡	AS_Register_Defaults (see page 36)	brief
≡	AS_Register_Name_To_Offset (see page 36)	brief
≡	AS_Registers_Read (see page 37)	brief
≡	AS_Registers_Write (see page 38)	brief
≡	AS_Time_Statistics (see page 39)	
≡	Bank_Name_To_Symbol (see page 40)	This is function Bank_Name_To_Symbol.
≡	Bank_Symbol_To_Name (see page 41)	This is function Bank_Symbol_To_Name.
≡	Bit_String_Index_By_Name (see page 41)	brief
≡	Bit_String_Report (see page 42)	brief
≡	Buffer_Init (see page 43)	brief
≡	Buffer_Length (see page 43)	brief
≡	Buffer_Maximum_Int32 (see page 44)	brief
≡	Buffer_Mean_Int32 (see page 44)	brief
≡	Buffer_Minimum_Int32 (see page 45)	brief
≡	Buffer_Peak_To_Peak_Int32 (see page 45)	brief
≡	Buffer_Save (see page 46)	brief
≡	Buffer_Save_Binary_Int32 (see page 46)	brief
≡	Buffer_Standard_Deviation_Int32 (see page 47)	Uses Bessel's correction.
≡	Buffer_Stuff (see page 48)	brief

Character_Get (see page 48)	brief Obtains a key from the keyboard in a non-blocking way. This is exerpetted from AES Universal Library/Driver. Excerpt from AES advanced Linux library/driver. COPYRIGHT NOTICE Copyright (c) 2012 by Apex Embedded Systems. param[out] character Character from keyboard otherwise null character.
CMD__AS_Calibration_Gain (see page 49)	brief
CMD__AS_Calibration_Offset (see page 50)	brief
CMD__AS_Gancal (see page 51)	brief
CMD__AS_OfsCAL (see page 51)	brief
CMD__AS_Parameter (see page 52)	brief
CMD__AS_Rdata (see page 54)	brief In order to use Data Read in Stop Continuous Mode one must issue the following command sequence: b idi spi mode 1 b idi spi clk 100000 b idi as sdatac b idi as wreg 0 0x85 0x90 0x00 0x00 0x00 0x00 0x00 0x00 0x40 b idi as rreg 0 9 b idi as start b idi as rdata 1024 --file out.txt
CMD__AS_Rdatac (see page 58)	brief
CMD__AS_Reg_Write_By_Name (see page 58)	brief
CMD__AS_Register_Load (see page 60)	brief
CMD__AS_Register_Save (see page 61)	brief
CMD__AS_Reset (see page 61)	brief
CMD__AS_RReg (see page 62)	brief
CMD__AS_Sdatac (see page 64)	brief
CMD__AS_Sleep (see page 64)	brief
CMD__AS_Start (see page 65)	brief
CMD__AS_Stop (see page 65)	brief
CMD__AS_Wakeup (see page 66)	brief
CMD__AS_WReg (see page 66)	brief
CMD__FRAM_Dump (see page 67)	brief
CMD__FRAM_Init (see page 68)	brief
CMD__FRAM_Load (see page 69)	brief
CMD__FRAM_RDID (see page 70)	brief
CMD__FRAM_RDSR (see page 70)	brief
CMD__FRAM_Save (see page 71)	brief
CMD__FRAM_WRDI (see page 72)	brief
CMD__FRAM_WREN (see page 72)	brief
CMD__FRAM_Write (see page 73)	brief
CMD__FRAM_WRSR (see page 73)	brief
CMD__IAI16_AI_All (see page 74)	brief
CMD__IAI16_AI_Channel (see page 75)	brief
CMD__IAI16_AI_ID (see page 75)	brief
CMD__IDI48_DIN_All (see page 76)	brief
CMD__IDI48_DIN_Channel (see page 77)	brief
CMD__IDI48_DIN_Group (see page 78)	brief
CMD__IDI48_DIN_Helper_All_Values_False (see page 79)	brief

≡	CMD_IDI48_DIN_ID (see page 79)	brief
≡	CMD_IDI48_DIN_Test (see page 80)	brief
≡	CMD_IDI48_DIN_Test_FE (see page 81)	brief
≡	CMD_IDI48_DIN_Test_Interrupt (see page 84)	brief Assumption: IRQs slower than main (see page 239) loop below.
≡	CMD_IDI48_DIN_Test_Interrupt_Handler (see page 89)	brief Interrupt Service Routine (ISR). This is a legacy implementation.
≡	CMD_IDI48_DIN_Test_RE (see page 90)	brief
≡	CMD_IDI48_DIN_Test_Value (see page 93)	brief
≡	CMD_IDO48_DO_All (see page 96)	This is function CMD_IDO48_DO_All.
≡	CMD_IDO48_DO_Channel (see page 97)	brief
≡	CMD_IDO48_DO_Group (see page 98)	brief
≡	CMD_IDO48_DO_ID (see page 99)	brief
≡	CMD_IDO48_DOUT_Test_Alternate (see page 100)	brief
≡	CMD_IDO48_DOUT_Test_One_Hot (see page 102)	brief
≡	CMD_IDO48_IDI48_Loopback (see page 103)	brief
≡	CMD_IDO48_Test (see page 109)	brief Cycles through all channels one at a time in a one-hot situation.
≡	CMD_Main_AnalogStick_CS (see page 110)	brief
≡	CMD_Main_Base (see page 110)	brief
≡	CMD_Main_FRAM_CS (see page 111)	brief
≡	CMD_Main_I_Count (see page 112)	brief Number of interrupt cycles. If "loop" is used, then the count can be terminated by pressing any key on the keyboard input.
≡	CMD_Main_Init_Reg (see page 113)	brief
≡	CMD_Main_IO_Behavior (see page 114)	brief
≡	CMD_Main_Irq_Number (see page 115)	brief
≡	CMD_Main_Mode_Jumpers (see page 116)	brief Mode jumper settings. 0 = 00 =
≡	CMD_Main_Trace (see page 117)	brief
≡	CMD_SPI_Commit (see page 118)	brief
≡	CMD_SPI_Config_Chip_Select_Behavior (see page 118)	brief
≡	CMD_SPI_Config_Chip_Select_Route (see page 120)	brief
≡	CMD_SPI_Config_Clock_Hz (see page 120)	brief
≡	CMD_SPI_Config_End_Cycle_Delay_Sec (see page 121)	brief
≡	CMD_SPI_Config_Get (see page 122)	brief
≡	CMD_SPI_Config_Mode (see page 123)	CPOL CPHA MODE 0 0 0 1 0 1 2 1 0 3 1 1
≡	CMD_SPI_Config_SDIL_Polarity (see page 124)	brief
≡	CMD_SPI_Config_SDIO_Wrap (see page 125)	brief
≡	CMD_SPI_Config_SDO_Polarity (see page 126)	brief
≡	CMD_SPI_Data (see page 126)	brief

≡	CMD__SPI_Data_Interpreter (see page 128)	brief
≡	CMD__SPI_FIFO (see page 129)	brief
≡	CMD__SPI_ID (see page 132)	brief
≡	CMD__SPI_Status (see page 132)	brief
≡	CMD__Trace_File (see page 133)	brief
≡	CMD__Trace_Start (see page 134)	brief
≡	CMD__Trace_Stop (see page 135)	brief
≡	Command_Line_Analog_Stick (see page 136)	brief
≡	Command_Line_Dump (see page 137)	brief
≡	Command_Line_FPGA (see page 138)	brief
≡	Command_Line_FRAM (see page 139)	brief
≡	Command_Line_Help (see page 140)	brief
≡	Command_Line_IAI16 (see page 140)	brief
≡	Command_Line_IDI48 (see page 141)	brief
≡	Command_Line_IDO48 (see page 142)	brief
≡	Command_Line_IO_Read (see page 143)	brief used to read one or more values from registers sequentially. b [prefix commands] [x] [c]
≡	Command_Line_IO_Write (see page 145)	brief Used to write to one or more registers in sequence. Can write values to the same register by using the 'x' to toggle auto-indexing from initially on to the off position.
≡	Command_Line_Loop_Count (see page 146)	brief
≡	Command_Line_Loop_Delay (see page 147)	brief
≡	Command_Line_Loop_Space (see page 148)	brief
≡	Command_Line_Main (see page 148)	brief Processes and dispatches the top level of the command and passes the remaining string list onto specialized functions to further process arguments. If no command is specified then a help output is produced. param[in] argc number of arguments including the executable file name param[in] argv list of string arguments lex'd from the command line
≡	Command_Line_Main__Board_Type_Status (see page 149)	brief Determine if this command requires specification of board type or not, or if a board type can be implied.
≡	Command_Line_Prefix (see page 150)	brief
≡	Command_Line_Prefix_Irq (see page 151)	brief
≡	Command_Line_Prefix_Loop (see page 152)	brief
≡	Command_Line_Prefix_Trace (see page 153)	brief

.Command_Line_Register_Transaction (see page 154)	brief Either reads or writes a register using the form: id1 [] If is not include, then it is assumed to be a read. If value is included, then a write to the specified register is made. This function uses the definitions[] array which is global and built from IDI_REGISTER_SET_DEFINITION (see page 350) macro which is a nicely organized register list. param[in] argc number of arguments including the executable file name param[in] argv list of string arguments lex'd from the command line
.Command_Line_Set (see page 156)	brief
.Command_Line_SPI (see page 157)	brief
.Command_Line_Wait (see page 158)	brief
.EC_Code_To_Human_Readable (see page 158)	This is function EC_Code_To_Human_Readable.
.FPGA_Date_Time_Information (see page 159)	brief
.FRAM_Chip_Select_Route_Override (see page 161)	brief FRAM SPI chip select over-ride mechanism. Temporarily used to set the chip select channel to what we need based on "set frams".
.FRAM_Chip_Select_Route_Restore (see page 162)	brief FRAM SPI chip select restore mechanism. Returns the chip select to original SPI configuration settings.
.FRAM_Memory_Read (see page 162)	brief Reads data from FRAM memory to the output buffer (see page 325). param[in] address Starting FRAM address param[in] count Number of bytes to transfer param[out] buffer (see page 325) Destination buffer (see page 325) in which to store the data
.FRAM_Memory_Write (see page 164)	brief Writes data from buffer (see page 325) to FRAM memory. param[in] address Starting FRAM address param[in] count Number of bytes to transfer param[out] buffer (see page 325) Source buffer (see page 325) from which data will be transferred to FRAM
.FRAM_Read_ID (see page 165)	brief param[out] id The 32-bit ID register read from the FRAM. This appears to be only available with Fujitsu parts.
.FRAM_Read_Status_Register (see page 166)	brief Read the FRAM status register and output the value.
.FRAM_Write_Disable (see page 167)	brief FRAM Write Latch disable (or clear) command (WRDI).
.FRAM_Write_Enable_Latch_Set (see page 168)	brief FRAM Write Enable Latch Set command (WREN)
.FRAM_Write_Status_Register (see page 169)	brief Write to the FRAM status register. param[in] FRAM status value to be written.
.FRAM_File_To_Memory (see page 170)	brief
.FRAM_Memory_To_File (see page 171)	brief
.FRAM_Report (see page 172)	brief

FRAM_Set (see page 173)	This function will be used when creating a memory pool so that as blocks are allocated one can determine if we have an issue outside of any allocated space (i.e. overflows and so on). param[in] cfg pass in the configuration to be written to hardware. return a nonzero if successful, else return zero.
Help (see page 175)	brief Outputs a help listing to the user.
Help_Output (see page 176)	brief
Help_Pause_Helper (see page 177)	brief Outputs a help listing to the user.
Hex_Dump_Line (see page 178)	This is function Hex_Dump_Line.
IAI_AI_ID_Get (see page 179)	brief Obtains the analog input component (or board ID in this case) ID number. param[out] id The 16-bit ID number
IAI16_AI_Channel_Get (see page 179)	brief
IAI16_AI_Channel_Multiple_Get (see page 180)	brief
IDI_Bank_Name_To_Symbol (see page 181)	brief param[in] bank the bank enumerated value written to the bank register.
IDI_Bank_Symbol_To_Name (see page 181)	brief Outputs a human readable CSV to the desired output file or stdout. param[in] bank the bank enumerated value written to the bank register.
IDI_Dataset_Defaults (see page 182)	brief
IDI_DIN_Channel_Get (see page 183)	brief Obtains and reports a single digital input channel. param[in] channel channel to be read out. param[out] value Pointer to the boolean value to be set based on the digital input value
IDI_DIN_Group_Get (see page 183)	brief Reads the selected digital input port (8-bits). param[in] group the group, range is 0 to 5. param[out] value pointer to the destination for the data read out
IDI_DIN_ID_Get (see page 184)	brief Obtains the digital input component (or board ID in this case) ID number. param[out] id The 16-bit ID number
IDI_DIN_IsNotPresent (see page 185)	brief Determines if the DIN component and/or board is present. Returns true if not present (i.e. error).
IDI_DIN_Pending_Clear (see page 185)	brief param[in] channel channel to be cleared of its pending status. param[out] value pointer to the destination for the data read out
IDI_Help (see page 186)	brief
IDI_Initialization_Data_Structure_Load (see page 186)	brief
IDI_Initialization_Register (see page 187)	brief
IDI_Main_Loop_Testing (see page 188)	brief
IDI_Register_Report_CSV (see page 189)	brief
IDI_Termination (see page 190)	brief
IDI48_DIN_ID_Port_Scan (see page 191)	brief
IDO_Bank_Name_To_Symbol (see page 192)	This is function IDO_Bank_Name_To_Symbol.
IDO_Bank_Symbol_To_Name (see page 193)	This is function IDO_Bank_Symbol_To_Name.

☰	IDO_Dataset_Defaults (see page 193)	brief
☰	IDO_DO_Channel_Get (see page 194)	brief Obtains and reports a single digital output channel. param[in] channel channel to be read out. param[out] value Pointer to the boolean value to be set based on the digital input value
☰	IDO_DO_Channel_Set (see page 195)	brief Sets a single digital output channel. param[in] channel to be written. param[in] value
☰	IDO_DO_Group_Get (see page 195)	brief Reads the selected digital output port (8-bits). param[in] group the group, range is 0 to 5. param[out] value pointer to the destination for the data read out
☰	IDO_DO_Group_Set (see page 196)	brief Reads the selected digital output port (8-bits). param[in] group the group, range is 0 to 5. param[in] value pointer to the destination for the data read out
☰	IDO_DOUT_ID_Get (see page 196)	brief Obtains the digital output component (or board ID in this case) ID number. param[out] id The 16-bit ID number
☰	IDO_DOUT_IsNotPresent (see page 197)	brief Determines if the DIN component and/or board is present. Returns true if not present (i.e. error).
☰	IDO_Help (see page 198)	brief
☰	IDO_Initialization_Data_Structure_Load (see page 198)	brief
☰	IDO_Initialization_Register (see page 199)	brief
☰	IDO_Register_Report_CSV (see page 200)	brief
☰	IDO_Termination (see page 201)	brief
☰	IDO48_IDI48_Loopback_Test (see page 202)	brief
☰	Initialization (see page 202)	brief Runs upon application startup. It restores the idi_dataset (see page 302) data structure or if the file cannot be found it will simply initialize those parameters to default values.
☰	IO_Direction_IsNotValid (see page 204)	brief Looks up in the register definitions list for the ports possible read/write directions. param[in] location the enumerated register symbol. The enumerated symbol is composed of offset and bank information used to determine the final address information. param[in] direction The desired direction that is to run
☰	IO_Get_Symbol_Name (see page 204)	brief Translates a register enumerated symbol into a string that is the same as the enumerated symbol used throughout this code base. param[in] location The enumerated symbol representing the register.
☰	IO_Read_U16_Address_Fixed (see page 205)	brief Reads uint16_t (see page 321) from I/O ports in a uint8_t (see page 321) succession to the same address location. Macros are used to guide the target implementation. In this case, bus width (which we typically refer to the port width, which is different than register width) is assumed to be byte (uint8_t (see page 321)) wide. param[in] location the enumerated register symbol. The enumerated symbol is composed of offset and bank information used to determine the final address information. param[in] value The pointer to the destination of the read uint16_t (see page 321) value.

☰	IO_Read_U16_Address_Increment (see page 206)	brief Reads uint16_t (see page 321) from I/O ports in a uint8_t (see page 321) succession incrementing the offset address. Macros are used to guide the target implementation. In this case, bus width (which we typically refer to the port width, which is different than register width) is assumed to be byte (uint8_t (see page 321)) wide. param[in] location the enumerated register symbol. The enumerated symbol is composed of offset and bank information used to determine the final address information. param[in] value The pointer to the destination of the read uint16_t (see page 321) value.
☰	IO_Read_U8 (see page 206)	brief Reads uint8_t (see page 321) from I/O port. Macros are used to guide the target implementation. param[in] location the enumerated register symbol. The enumerated symbol is composed of offset and bank information used to determine the final address information. param[in] value The pointer to the destination of the read uint8_t (see page 321) value.
☰	IO_Read_U8_Port (see page 208)	brief
☰	IO_Trace_Dump (see page 208)	brief Dumps the trace buffer (see page 325) to a CSV file for further analysis.
☰	IO_Trace_Initialize (see page 211)	brief
☰	IO_Trace_Log (see page 212)	brief
☰	IO_Trace_Log_Dataset (see page 212)	brief
☰	IO_Trace_Log_Dataset_Begin (see page 213)	brief
☰	IO_Trace_Log_Dataset_End (see page 214)	brief
☰	IO_Trace_Log_Function (see page 215)	brief
☰	IO_Write_U16_Address_Fixed (see page 216)	brief Writes uint16_t (see page 321) to I/O ports in a uint8_t (see page 321) succession to the same address location. Macros are used to guide the target implementation. In this case, bus width (which we typically refer to the port width, which is different than register width) is assumed to be byte (uint8_t (see page 321)) wide. param[in] location the enumerated register symbol. The enumerated symbol is composed of offset and bank information used to determine the final address information. param[in] value The uint16_t (see page 321) value to be written to the I/O register
☰	IO_Write_U16_Address_Increment (see page 216)	brief Writes uint16_t (see page 321) to I/O ports in a uint8_t (see page 321) succession incrementing the offset address. Macros are used to guide the target implementation. In this case, bus width (which we typically refer to the port width, which is different than register width) is assumed to be byte (uint8_t (see page 321)) wide. param[in] location the enumerated register symbol. The enumerated symbol is composed of offset and bank information used to determine the final address information. param[in] value The uint16_t (see page 321) value to be written to the I/O register

☰	IO_Write_U8 (see page 217)	brief Writes uint8_t (see page 321) to I/O port. Macros are used to guide the target implementation. param[in] board_id software level board selector or descriptor. param[in] location the enumerated register symbol. The enumerated symbol is composed of offset and bank information used to determine the final address information. param[in] value The data to be written out.
☰	IO_Write_U8_Port (see page 218)	brief
☰	IOKern_DOS_IRQ_Free (see page 219)	brief
☰	IOKern_DOS_IRQ_Mask_Get (see page 220)	
☰	IOKern_DOS_IRQ_Mask_Set (see page 220)	
☰	IOKern_DOS_IRQ_Request (see page 221)	brief
☰	IOKern_DOS_ISR_Install (see page 222)	brief
☰	IOKern_DOS_ISR_Restore (see page 223)	brief
☰	IOKern_DOS_Vector_Get (see page 224)	
☰	IOKern_DOS_Vector_Set (see page 225)	
☰	IOKern Interrupt_Helper (see page 225)	brief Supporting 'fast' interrupts
☰	IOKern_IRQ_Invoke_ISR_Test (see page 226)	brief
☰	IOKern_IRQ_Test (see page 226)	brief
☰	IOKern_IRQ_Test_Helper (see page 227)	brief
☰	IOKern_ISR0 (see page 228)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
☰	IOKern_ISR1 (see page 228)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
☰	IOKern_ISR10 (see page 229)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
☰	IOKern_ISR11 (see page 229)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
☰	IOKern_ISR12 (see page 230)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
☰	IOKern_ISR13 (see page 230)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
☰	IOKern_ISR14 (see page 231)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
☰	IOKern_ISR15 (see page 231)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
☰	IOKern_ISR2 (see page 232)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
☰	IOKern_ISR3 (see page 232)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.

☰	IOKern_ISR4 (🔗 see page 233)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
☰	IOKern_ISR5 (🔗 see page 233)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
☰	IOKern_ISR6 (🔗 see page 234)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
☰	IOKern_ISR7 (🔗 see page 234)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
☰	IOKern_ISR8 (🔗 see page 235)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
☰	IOKern_ISR9 (🔗 see page 235)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
☰	IOKern_PIC_EOI (🔗 see page 236)	brief
☰	IOKern_Resource_Initialization (🔗 see page 236)	
☰	IOKern_Resource_Termination (🔗 see page 237)	
☰	IOKern_Task_Handle_Get (🔗 see page 238)	brief
☰	IOKern_Task_ID_Get (🔗 see page 238)	brief
☰	main (🔗 see page 239)	brief Processes and dispatches the top level of the command and passes the remaining string list onto specialized functions to further process arguments. If no command is specified then a help output is produced. param[in] argc number of arguments including the executable file name param[in] argv list of string arguments lex'd from the command line
☰	Main_Versalogic (🔗 see page 243)	brief
☰	Print_Byt_List (🔗 see page 243)	brief
☰	Print_Multiple (🔗 see page 245)	brief
☰	Register_Acronym_To_Row (🔗 see page 245)	brief
☰	Register_Report_CSV (🔗 see page 246)	brief Outputs a human readable CSV to the desired output file or stdout. param[in] table register definition table or data structure array param[in] out destination of the human readable text to specified file or terminal.
☰	Signal_Handler (🔗 see page 247)	brief
☰	SPI_Calculate_Clock (🔗 see page 247)	brief Computes the SPI clock half clock register value given a requested SPI clock frequency. It will also produce a 'report' indicating the actual value (due to integer resolution) as well as a computed error between requested and actual. The eror can be used to determine whether timing constraints are met. param[in] clock_request_hz Request clock frequency in Hertz. Example: 1.0e6 is 1MHz. param[in] clock_actual_hz Actual computed frequency. If this pointer is NULL, then it is not output. param[out] error Error between requested and actual. If this pointer is NULL, then it is not output. param[out] hci Half clock register value... more (🔗 see page 247)

✳️	SPI_Calculate_End_Cycle_Delay (🔗 see page 248)	brief Computes the time delay at the end of each byte transmitted. It will only output the parameters whose pointers are not NULL. param[in] spi_half_clock_interval_sec Computed half clock interval in seconds param[in] delay_request_sec Requested time delay in seconds param[out] delay_actual_sec Pointer to actual time delay computed, if not NULL. param[out] error Pointer to error value computed, if not NULL. param[out] ecd Pointer to the computed end-cycle-delay, if not NULL.
✳️	SPI_Calculate_Half_Clock (🔗 see page 249)	brief Computes the half clock register value given a requested time interval. It will also produce a 'report' indicating the actual value (due to integer resolution) as well as a computed error between requested and actual. The error can be used to determine whether timing constraints are met. param[in] half_clock_request_sec Request time interval in seconds. Example: 20.0e-6 is 20uS. param[in] half_clock_actual_sec Actual computed time. If this pointer is NULL, then it is not output. param[out] error Error between requested and actual. If this pointer is NULL, then it is not output. param[out] hci Half clock register value computed. If this... more (🔗 see page 249)
✳️	SPI_Calculate_Half_Clock_Interval_Sec (🔗 see page 251)	brief Computes the half clock interval in seconds given the value from the half clock interval register. param[in] half_clock_interval Half clock interval register value
✳️	SPI_Commit (🔗 see page 251)	brief Sets/Clears the chip select or used to commit the transmit/write FIFO to the spi interface. The mode of operation is dependent on the chip_select_behavior. param[in] commit Used to write to the SCS_COMMIT bit. Its behavior is dependent on the chip_select_behavior.
✳️	SPI_Commit_Get (🔗 see page 252)	brief Sets/Clears the chip select or used to commit the transmit/write FIFO to the spi interface. The mode of operation is dependent on the chip_select_behavior. param[out] commit Used to write to the SCS_COMMIT bit. Its behavior is dependent on the chip_select_behavior.
✳️	SPI_Commit_Is_Inactive (🔗 see page 253)	brief
✳️	SPI_Configuration_Chip_Select_Behavior_Get (🔗 see page 253)	brief Extracts the chip select behavior from the SPI configuration register. param[out] chip_select_behavior pointer to the destination of the value obtained.
✳️	SPI_Configuration_Chip_Select_Behavior_Set (🔗 see page 254)	brief Sets the chip select behavior to the SPI configuration register. param[in] chip_select_behavior enumerated value to be written to the register.
✳️	SPI_Configuration_Chip_Select_Route_Get (🔗 see page 255)	brief Sets which hardware chip select line is to be used, either CS0 (channel 0) or CS1 (channel 1). param[out] chip_select_behavior pointer to the destination of the value obtained.
✳️	SPI_Configuration_Chip_Select_Route_Set (🔗 see page 256)	brief Sets which hardware chip select line is to be used, either CS0 (channel 0) or CS1 (channel 1). param[in] cs_to_channel1 true if CS to route to channel 1 (CS1), otherwise channel 0 (CS0).
✳️	SPI_Configuration_Get (🔗 see page 257)	brief Obtains the SPI configuration from the hardware. param[out] cfg SPI configuration data structure or data set

✳️	SPI_Configuration_Initialize (see page 258)	brief Initializes the SPI configuration data structure param[in] cfg Pointer to the SPI configuration data structure to be initialized
✳️	SPI_Configuration_Set (see page 259)	brief Commits the configuration data structure to the hardware. param[in] cfg The software configuration data structure to be committed to hardware
✳️	SPI_Data_Read (see page 261)	Special case of Write/Read that has a function signature same as fread() or fwrite(). param[in] cfg pass in the configuration to be written to hardware. return a nonzero if successful, else return zero.
✳️	SPI_Data_Write (see page 262)	Special case of Write/Read that has a function signature same as fread() or fwrite(). param[in] cfg pass in the configuration to be written to hardware. return a nonzero if successful, else return zero.
✳️	SPI_Data_Write_Read (see page 263)	brief If CSB = software. Stuff as fast as you can. If CSB == buffer (see page 325). if object size = 1, then stuff as fast as you can. The chip select will go active and only go inactive when done transmitting. This is best used for objects whos size is less than SPI_FIFO_SIZE. if object size != 1, then stuff only that quantity and wait until buffers are empty in order to have the chip select wrap around the data. If CSB == uint8_t (see page 321) Stuff as fast as you can. If CSB == uint16_t (see page 321) Stuff two bytes at a time... more (see page 263)
✳️	SPI_Data_Write_Read_Helper_Commit (see page 270)	brief returns true if user wishes to break.
✳️	SPI_Data_Write_Read_Helper_Read (see page 271)	brief
✳️	SPI_Data_Write_Read_Helper_Write (see page 273)	brief
✳️	SPI_FIFO_Read (see page 276)	brief Reads from the SPI receive/read data FIFO. It does not attempt to correlate the number of transmit bytes with receive bytes. Its purpose is more for low level hardware testing. Note that this function has a signature identical to the fread() function (i.e. make use of function pointers to guide sourcing of data). param[in] buffer (see page 325) Buffer for the data destination. param[in] size Size of objects in bytes. param[in] count Number of objects to be read param[out] fd_log Optional log file to write the buffer (see page 325) too. If NULL, then no logging.
✳️	SPI_FIFO_Write (see page 278)	brief Writes specifically to the SPI transmit/write data FIFO. It does not attempt to correlate the number of transmit bytes with receive bytes. Its purpose is more for low level hardware testing. Note that this function has a signature identical to the fwrite() function (i.e. make use of function pointers to guide destination of data). param[in] buffer (see page 325) Buffer containing the data to be written. param[in] size Size of objects in bytes. param[in] count Number of objects to be written param[out] fd_log Optional log file to write the buffer (see page 325) too. If NULL, then no logging.

	SPI_ID_Get (see page 279)	This is function SPI_ID_Get.
	SPI_ID_Get_Helper (see page 280)	brief Retrieves the SPI ID register value. param[out] id The SPI component ID value.
	SPI_IsNotPresent (see page 280)	brief Reports if the SPI component is available within the register space by matching a known ID. The SPI register map is only enabled within the hardware if the hardware mode is not zero (i.e. M1 and M0 jumpers on the board provide a nonzero value).
	SPI_Report_Configuration_Text (see page 281)	brief Creates a human readable report of the SPI configuration data structure. param[in] cfg SPI configuration data structure pointer param[out] out File destination descriptor
	SPI_Report_Status_Text (see page 282)	brief Produces a human readable report of the SPI status data structure. param[in] status SPI status data structure pointer param[out] out File destination descriptor
	SPI_Status_Read (see page 283)	brief Builds a detailed status data structure of the receive/read incoming SPI data FIFO. Reports the quantity of bytes currently in the receive FIFO, full flag, empty flag, the total size of the FIFO in bytes, and sets tx_status to false indicating that this is status specific to the receive FIFO. The status register has the following format: status[7] full status[6] empty status[5] not used (future size expansion) status[4:0] number of bytes currently in the FIFO param[out] status Pointer to status data structure to be updated.
	SPI_Status_Read_FIFO_Is_Not_Empty (see page 284)	brief Returns the receive/read FIFO empty status flag. This function is typically used to determine if the FIFO is empty.
	SPI_Status_Read_FIFO_Status (see page 284)	brief Returns the complete read/receive FIFO status. The status register has the following format: status[7] full status[6] empty status[5] not used (future size expansion) status[4:0] number of bytes currently in the FIFO param[out] empty FIFO empty flag param[out] bytes_available a count of the number of bytes in the FIFO
	SPI_Status_Write (see page 286)	brief Builds a detailed status data structure of the transmit/write outgoing SPI data FIFO. Reports the quantity of bytes currently in the transmit FIFO, full flag, empty flag, the total size of the FIFO in bytes, and sets tx_status to true indicating that this is status specific to the transmit FIFO. The status register has the following format: status[7] full status[6] empty status[5] not used (future size expansion) status[4:0] number of bytes currently in the FIFO param[out] status Pointer to status data structure to be updated.
	SPI_Status_Write_FIFO_Is_Full (see page 287)	brief Returns the transmit/write FIFO full status flag. It is preferable to use the SPI_Status_Write (see page 286)() or SPI_Status_Write_FIFO_Status (see page 288)() because all status is retrieved at one time.
	SPI_Status_Write_FIFO_Is_Not_Empty (see page 288)	brief Returns the transmit/write FIFO empty status flag. This function is typically used to wait for the transmit/write FIFO to become empty.

	SPI_Status_Write_FIFO_Status (see page 288)	brief Returns the complete write/transmit FIFO status. The status register has the following format: status[7] full status[6] empty status[5] not used (future size expansion) status[4:0] number of bytes currently in the FIFO param[out] full FIFO full flag param[out] empty FIFO empty flag param[out] bytes_in_fifo a count of the number of bytes in the FIFO
	StopWatch_Close (see page 290)	This is function StopWatch_Close.
	StopWatch_Initialization (see page 291)	This is function StopWatch_Initialization.
	StopWatch_Open (see page 291)	This is function StopWatch_Open.
	StopWatch_Process (see page 291)	This is function StopWatch_Process.
	StopWatch_Set (see page 291)	This is function StopWatch_Set.
	StopWatch_Termination (see page 292)	This is function StopWatch_Termination.
	StopWatch_TimeStamp_String (see page 292)	This is function StopWatch_TimeStamp_String.
	StopWatch_Value (see page 292)	This is function StopWatch_Value.
	String_To_Bool (see page 293)	brief General function used to convert a string into a boolean equivalent value. param[in] str string input for conversion.
	Termination (see page 293)	brief Runs upon application exit. It saves the idi_dataset (see page 302) data structure.
	Time_Current_String (see page 294)	brief Obtains the current time and date. This is exerpetted from AES Universal Library/Driver. Excerpt from AES advanced Linux library/driver. COPYRIGHT NOTICE Copyright (c) 2012 by Apex Embedded Systems. param[out] buf resulting date-time string. param[in] buf_size the character size of the buffer (see page 325) including null terminating character.

Legend

	Method
---	--------

3.2.1.1 ADS1259_Calibration_Range_Test Function

C++

```
static int ADS1259_Calibration_Range_Test(int32_t * value, size_t address, size_t
clamp_and_error_skip);
```

File

idi.c (see page 373)

Description

brief

Body Source

```
1: static int ADS1259_Calibration_Range_Test( int32_t * value, size_t address, size_t
clamp_and_error_skip )
2: {
```

```

3: int32_t scratch;
4: int error_code = SUCCESS;
5:
6: scratch = *value;
7: if ( AS_AM2315_OF0 == address )
8: {
9:     if ( scratch > ADS1259_OFC_MAX )
10:    {
11:        error_code = -EC_RANGE;
12:        scratch = ADS1259_OFC_MAX;
13:    }
14:    if ( scratch < ADS1259_OFC_MIN )
15:    {
16:        error_code = -EC_RANGE;
17:        scratch = ADS1259_OFC_MIN;
18:    }
19: }
20: else
21: {
22:     if ( scratch > ADS1259_FSC_MAX )
23:    {
24:        error_code = -EC_RANGE;
25:        scratch = ADS1259_FSC_MAX;
26:    }
27:     if ( scratch < ADS1259_FSC_MIN )
28:    {
29:        error_code = -EC_RANGE;
30:        scratch = ADS1259_FSC_MIN;
31:    }
32: }
33:
34: if ( 0 != clamp_and_error_skip )
35: {
36:     *value      = scratch;
37:     error_code = SUCCESS;
38: }
39: return error_code;
40: }
```

3.2.1.2 AS_Calibration_Value_Read Function

C++

```
static int AS_Calibration_Value_Read(int board_id, size_t address, int32_t * value);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int AS_Calibration_Value_Read( int board_id, size_t address, int32_t * value )
2: {
3:     int error_code;
4:     uint32_t value_temp;
5:     size_t index;
6:     const size_t byte_length = 3;
```

```

7:  uint8_t  tx_buf[12] = { 0x00 };
8:  uint8_t  rx_buf[12] = { 0x00 };
9:
10: #define FUNCTION_NAME__ "AS_Calibration_Value_Read"
11: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
12: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
13: #endif
14: #undef FUNCTION_NAME__
15:
16: error_code = AS_Registers_Read( board_id, address, byte_length, tx_buf, rx_buf );
17: if ( error_code < 0 ) goto AS_Calibration_Value_Read_Error_Exit;
18: value_temp = 0;
19:
20: for ( index = 0; index < byte_length; index++ ) value_temp = ( value_temp << 8 ) + ( (uint32_t) rx_buf[byte_length - index + 2 - 1] );
21:
22:
23: if ( 0 != ( 0xFF800000L & value_temp ) )
24: {
25:   value_temp = 0xFF800000L | value_temp;
26: }
27: *value = (int32_t) value_temp;
28: return SUCCESS;
29: AS_Calibration_Value_Read_Error_Exit:
30: return error_code;
31: }
```

3.2.1.3 AS_Calibration_Value_Write Function

C++

```
static int AS_Calibration_Value_Write(int board_id, size_t address, int32_t value);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int AS_Calibration_Value_Write( int board_id, size_t address, int32_t value )
2: {
3:   int     error_code;
4:   uint32_t scratch;
5:   size_t   index;
6:   const size_t byte_length = 3;
7:   uint8_t  tx_buf[12] = { 0x00 };
8:   uint8_t  rx_buf[12] = { 0x00 };
9:
10: #define FUNCTION_NAME__ "AS_Calibration_Value_Write"
11: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
12: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
13: #endif
14: #undef FUNCTION_NAME__
15:
16:
17: ADS1259_Calibration_Range_Test( &value, address, 1 );
18:
```

```

19: scratch = (uint32_t) value;
20: for ( index = 0; index < byte_length; index++ )
21: {
22:     tx_buf[index + 2] = (uint8_t)( scratch & 0xFF );
23:     scratch = scratch >> 8;
24: }
25:
26: error_code = AS_Registers_Write( board_id, address, byte_length, tx_buf, rx_buf );
27: return error_code;
28: }
```

3.2.1.4 AS_Chip_Select_Route_Override Function

C++

```
static int AS_Chip_Select_Route_Override(int board_id);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Analog Stick SPI chip select over-ride mechanism. Temporarily used to set the chip select channel to what we need based on "set framcs".

Body Source

```

1: static int AS_Chip_Select_Route_Override( int board_id )
2: {
3:     struct board_dataset * dataset;
4:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
5:
6: #define FUNCTION_NAME__ "AS_Chip_Select_Route_Override"
7: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
8:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
9: #endif
10: #undef FUNCTION_NAME__
11:
12: SPI_Configuration_Chip_Select_Route_Get( board_id, &(dataset->as_spi_cs_route_backup) );
13: SPI_Configuration_Chip_Select_Route_Set( board_id, dataset->as_cs_default );
14: return SUCCESS;
15: }
```

3.2.1.5 AS_Chip_Select_Route_Restore Function

C++

```
static int AS_Chip_Select_Route_Restore(int board_id);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Analog Stick SPI chip select restore mechanism. Returns the chip select to original SPI configuration settings.

Body Source

```

1: static int AS_Chip_Select_Route_Restore( int board_id )
2: {
3:     struct board_dataset * dataset;
4:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
5:
6: #define FUNCTION_NAME__ "AS_Chip_Select_Route_Restore"
7: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
8:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
9: #endif
10: #undef FUNCTION_NAME__
11:
12: SPI_Configuration_Chip_Select_Route_Set( board_id, dataset->as_spi_cs_route_backup );
13: return SUCCESS;
14: }
```

3.2.1.6 AS_Data_Read Function

C++

```
static int AS_Data_Read(int board_id, int32_t * adc_value);
```

File

idi.c (see page 373)

Description

brief Rdata command

Body Source

```

1: static int AS_Data_Read( int board_id, int32_t * adc_value )
2: {
3:     int error_code;
4:     uint32_t adc_temp;
5:     size_t byte_index;
6:     uint8_t tx_buf[4] = { 0x12, 0x00, 0x00, 0x00 };
7:     uint8_t rx_buf[4] = { 0x00, 0x00, 0x00, 0x00 };
8:
9:
10:    AS_Chip_Select_Route_Override( board_id );
11:    SPI_Configuration_Chip_Select_Behavior_Set( board_id, CSB_BUFFER );
12:    error_code = SPI_Data_Write_Read( board_id,
13:                                     1,
14:                                     4,
15:                                     tx_buf,
16:                                     4,
17:                                     rx_buf );
```

```

18:         );
19:     AS_Chip_Select_Route_Restore( board_id );
20:     if ( error_code < 0 ) goto AS_Read_Data_Exit;
21:
22:     adc_temp = 0;
23:     for ( byte_index = 1; byte_index < 4; byte_index++ ) adc_temp = (adc_temp << 8) + (
24:     (uint32_t) rx_buf[byte_index] );
25:
26:     if ( 0 != ( 0xFF800000L & adc_temp ) )
27:     {
28:         adc_temp = 0xFF800000L | adc_temp;
29:     }
30:     *adc_value = (int32_t) adc_temp;
31:
32:     AS_Read_Data_Exit:
33:     return error_code;
34: }
```

3.2.1.7 AS_Data_Ready_NoWait Function

C++

```
static int AS_Data_Ready_NoWait(int board_id, size_t drdy_anticipated_active);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int AS_Data_Ready_NoWait( int board_id, size_t drdy_anticipated_active )
2: {
3:     uint8_t tx_buf[4] = { 0x00, 0x00, 0x00, 0x00 };
4:     uint8_t rx_buf[4] = { 0x00, 0x00, 0x00, 0x00 };
5:     uint8_t scratch;
6:
7:     if ( 0 == drdy_anticipated_active ) scratch = 0x00;
8:     else                         scratch = 0x80;
9:
10:    rx_buf[2] = scratch;
11:    AS_Registers_Read( board_id, 2, 1, tx_buf, rx_buf );
12:
13:    if ( scratch == ( rx_buf[2] & 0x80 ) ) return 0;
14:
15:    return 1;
16: }
```

3.2.1.8 AS_Data_Ready_Wait Function

C++

```
static void AS_Data_Ready_Wait(int board_id, size_t drdy_anticipated_active, size_t * loop_count_per_data_ready);
```

File

idi.c (see page 373)

Description

brief

Todo

need a timeout mechanism.

Body Source

```

1: static void AS_Data_Ready_Wait( int board_id, size_t drdy_anticipated_active, size_t *
loop_count_per_data_ready )
2: {
3:     size_t test_count;
4:     uint8_t tx_buf[4] = { 0x00, 0x00, 0x00, 0x00 };
5:     uint8_t rx_buf[4] = { 0x00, 0x00, 0x00, 0x00 };
6:     uint8_t scratch;
7:
8:     if ( 0 == drdy_anticipated_active ) scratch = 0x00;
9:     else                                scratch = 0x80;
10:
11:    test_count = 0;
12:    rx_buf[2] = scratch;
13:    while ( scratch == ( rx_buf[2] & 0x80 ) )
14:    {
15:        AS_Registers_Read( board_id, 2, 1, tx_buf, rx_buf );
16:        test_count++;
17:    }
18:    if ( test_count > 1 )
19:    {
20:        if ( NULL != loop_count_per_data_ready )
21:        {
22:            *loop_count_per_data_ready = 0;
23:        }
24:    }
25: }
```

3.2.1.9 AS_Opcode_Write Function

C++

```
int AS_Opcode_Write(int board_id, uint8_t * tx_buf, size_t count);
```

File

idi.c (see page 373)

Description

brief

Body Source

```
1: int AS_Opcode_Write( int board_id, uint8_t * tx_buf, size_t count )
```

```

2: {
3:     int error_code;
4:
5:
6: #define FUNCTION_NAME__ "AS_Opcode_Write"
7: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
8:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
9: #endif
10: #undef FUNCTION_NAME__
11:
12: AS_Chip_Select_Route_Override( board_id );
13: SPI_Configuration_Chip_Select_Behavior_Set( board_id, CSB_BUFFER );
14: error_code = SPI_Data_Write_Read( board_id,
15:                                     1,
16:                                     count,
17:                                     tx_buf,
18:                                     0,
19:                                     NULL
20:                                     );
21: AS_Chip_Select_Route_Restore( board_id );
22: return error_code;
23: }
```

3.2.1.10 AS_Rdata_Statistics Function

C++

```
static int AS_Rdata_Statistics(FILE * out);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int AS_Rdata_Statistics( FILE * out )
2: {
3:     int error_code;
4:     size_t length;
5:     double mean;
6:     double maximum;
7:     double minimum;
8:     double peak_to_peak;
9:     double stdev;
10:
11:    Buffer_Length( &length );
12:
13:    Buffer_Mean_Int32( buffer, length, &mean );
14:    Buffer_Minimum_Int32( buffer, length, &minimum );
15:    Buffer_Maximum_Int32( buffer, length, &maximum );
16:    Buffer_Peak_To_Peak_Int32( minimum, maximum, &peak_to_peak );
17:    error_code = Buffer_Standard_Deviation_Int32( buffer, length, mean, &stdev );
18:    fprintf( out, "Data Statistics:\n" );
19:    fprintf( out, " sample_quantity      = %7d\n", (int) length );
20:    fprintf( out, " mean                  = %7.3f\n", mean );
21:    if ( SUCCESS == error_code )
22:    {
```

```

23:   fprintf( out, "  stdev          = %7.3f\n", stdev      ) ;
24: }
25:   fprintf( out, "  minimum        = %7.3f\n", minimum    ) ;
26:   fprintf( out, "  maximum        = %7.3f\n", maximum    ) ;
27:   fprintf( out, "  peak-to-peak  = %7.3f\n", peak_to_peak );
28:
29:   fprintf( out, "\n"           );
30:   return SUCCESS;
31: }
```

3.2.1.11 AS_Register_Defaults Function

C++

```
static void AS_Register_Defaults(uint8_t * buf, size_t count);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static void AS_Register_Defaults( uint8_t * buf, size_t count )
2: {
3:   if ( count < AS_ADS1259_REGISTER_QTY ) return;
4:   buf[AS_ADS1259_CONFIG0] = 0x85;
5:   buf[AS_ADS1259_CONFIG1] = 0x10;
6:   buf[AS_ADS1259_CONFIG2] = 0x00;
7:   buf[AS_ADS1259_OF0C0] = 0x00;
8:   buf[AS_ADS1259_OF0C1] = 0x00;
9:   buf[AS_ADS1259_OF0C2] = 0x00;
10:  buf[AS_ADS1259_FSC0] = 0x00;
11:  buf[AS_ADS1259_FSC1] = 0x00;
12:  buf[AS_ADS1259_FSC2] = 0x40;
13: }
```

3.2.1.12 AS_Register_Name_To_Offset Function

C++

```
int AS_Register_Name_To_Offset(char * name);
```

File

idi.c (see page 373)

Description

brief

Body Source

```
1: int AS_Register_Name_To_Offset( char * name )
```

```

2: {
3:   size_t index;
4:
5:   index = 0;
6:   while ( NULL != as_register_name[index] )
7:   {
8:     if ( 0 == strncmp( as_register_name[index], name ) )
9:     {
10:      return index;
11:    }
12:    index++;
13:  }
14: return -EC_NOT_FOUND;
15: }
```

3.2.1.13 AS_Registers_Read Function

C++

```
static int AS_Registers_Read(int board_id, size_t address, size_t count, uint8_t * tx_buf,
uint8_t * rx_buf);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int AS_Registers_Read( int board_id, size_t address, size_t count, uint8_t * 
tx_buf, uint8_t * rx_buf )
2: {
3:   size_t object_count;
4:   int error_code;
5:   struct board_dataset * dataset;
6:
7:   dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
8: #define FUNCTION_NAME__ "AS_Registers_Read"
9: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
10: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
11: #endif
12: #undef FUNCTION_NAME__
13:
14:
15: if ( ( address + count ) > AS_AM2315_REGISTER_QTY ) return -EC_REGISTER_ADDRESS_EXCEEDED;
16:
17: tx_buf[0] = 0x20 + ( (uint8_t) ( 0x0F & address ) );
18: if ( 0 == count )
19: {
20:   tx_buf[1] = 0;
21:   object_count = 2;
22:   count = 1;
23: }
24: else
25: {
26:   tx_buf[1] = ( (uint8_t) ( 0x0F & count ) ) - 1;
27:   object_count = ( 0x0F & count ) + 2;
28: }
```

```

29:
30: if ( true == dataset->set__suppress_io_activity )
31: {
32:     size_t index;
33:     for ( index = 0; index < count; index++ )
34:     {
35:         rx_buf[index + 2] = dataset->ads1259.reg[address + index];
36:     }
37:     return SUCCESS;
38: }
39:
40:
41: AS_Chip_Select_Route_Override( board_id );
42: SPI_Configuration_Chip_Select_Behavior_Set( board_id, CSB_BUFFER );
43: error_code = SPI_Data_Write_Read( board_id,
44:                                     1,
45:                                     object_count,
46:                                     tx_buf,
47:                                     object_count,
48:                                     rx_buf
49:                                     );
50: AS_Chip_Select_Route_Restore( board_id );
51: return error_code;
52: }
```

3.2.1.14 AS_Registers_Write Function

C++

```
static int AS_Registers_Write(int board_id, size_t address, size_t count, uint8_t * tx_buf,
uint8_t * rx_buf);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int AS_Registers_Write( int board_id, size_t address, size_t count, uint8_t *
tx_buf, uint8_t * rx_buf )
2: {
3:     size_t      object_count;
4:     size_t      index;
5:     int         error_code;
6:     struct board_dataset * dataset;
7:
8:     error_code = SUCCESS;
9:     dataset    = ( struct board_dataset * ) board_definition[board_id].dataset;
10:
11: #define FUNCTION_NAME__ "AS_Registers_Write"
12: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
13:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
14: #endif
15: #undef FUNCTION_NAME__
16:
17:
18: if ( ( address + count ) > AS_ADS1259_REGISTER_QTY ) return -EC_REGISTER_ADDRESS_EXCEEDED;
```

```

19:
20:
21: tx_buf[0] = 0x40 + ( (uint8_t) ( 0x0F & address ) );
22: if ( 0 == count )
23: {
24:     tx_buf[1] = 0;
25:     count = 1;
26:     object_count = 3;
27: }
28: else
29: {
30:     tx_buf[1] = ( (uint8_t) ( 0x0F & count ) ) - 1;
31:     object_count = ( 0x0F & count ) + 2;
32: }
33:
34: if ( true == dataset->set__suppress_io_activity )
35: {
36:
37:     for ( index = 0; index < count; index++ )
38:     {
39:         dataset->ads1259.reg[index + address] = tx_buf[index + 2];
40:     }
41:     return SUCCESS;
42: }
43:
44:
45: AS_Chip_Select_Route_Override( board_id );
46: SPI_Configuration_Chip_Select_Behavior_Set( board_id, CSB_BUFFER );
47: error_code = SPI_Data_Write_Read( board_id,
48:                                     1,
49:                                     object_count,
50:                                     tx_buf,
51:                                     object_count,
52:                                     rx_buf
53:                                     );
54: AS_Chip_Select_Route_Restore( board_id );
55:
56: return error_code;
57: }
```

3.2.1.15 AS_Time_Statistics Function

C++

```
static void AS_Time_Statistics(STOPWATCH_HANDLE_TYPE * stopwatch_handle, int time_overall_index, int time_reading_index, int time_writing_index, FILE * out);
```

File

idi.c (see page 373)

Description

```
*****
*
• @brief
```

Body Source

```

1: static void AS_Time_Statistics( STOPWATCH_HANDLE_TYPE * stopwatch_handle,
2:                                int      time_overall_index,
3:                                int      time_reading_index,
4:                                int      time_writing_index,
5:                                FILE    *      out
6:                                )
7: {
8:     STOPWATCH_TIMEVAL_TYPE * time_value;
9:     double time_result[STOPWATCH_QUANTITY];
10:    size_t sample_count;
11:    double dscratch;
12:
13:    Buffer_Length( &sample_count );
14:
15:    fprintf( out, "Timing Statistics:\n" );
16:    time_value = StopWatch_Value( stopwatch_handle[time_overall_index] );
17:    time_result[time_overall_index] = ( (double) time_value->tv_sec ) + ( (double)
time_value->tv_usec ) * 0.000001;
18:    fprintf( out, " Total time           = %7.3f sec, 100%%\n",
time_result[time_overall_index] );
19:
20:    if ( 0.0 == time_result[time_overall_index] ) return;
21:
22:    time_value = StopWatch_Value( stopwatch_handle[time_reading_index] );
23:    time_result[time_reading_index] = ( (double) time_value->tv_sec ) + ( (double)
time_value->tv_usec ) * 0.000001;
24:    fprintf( out, " Read timing          = %7.3f sec, %5.1f%%\n",
time_result[time_reading_index], ( time_result[time_reading_index]* 100.0 /
time_result[time_overall_index] ) );
25:
26:    time_value = StopWatch_Value( stopwatch_handle[time_writing_index] );
27:    time_result[time_writing_index] = ( (double) time_value->tv_sec ) + ( (double)
time_value->tv_usec ) * 0.000001;
28:    fprintf( out, " To File(s) timing   = %7.3f sec, %5.1f%%\n",
time_result[time_writing_index], ( time_result[time_writing_index]* 100.0 /
time_result[time_overall_index] ) );
29:
30:    fprintf( out, " Total Samples        = %7d\n", (int) sample_count );
31:    dscratch = ( (double) sample_count ) / time_result[time_overall_index];
32:    fprintf( out, " Samples/Sec          = %7.2f Samples/Sec\n", dscratch );
33:    dscratch = 1000000.0 / dscratch;
34:    fprintf( out, " Time per sample     = %7.3f uSec/Sample\n", dscratch );
35: }
```

3.2.1.16 Bank_Name_To_Symbol Function

C++

```
static int Bank_Name_To_Symbol(int board_id, const char * name, int * bank);
```

File

idi.c (see page 373)

Description

This is function Bank_Name_To_Symbol.

Body Source

```

1: static int Bank_Name_To_Symbol( int board_id, const char * name, int * bank )
2: {
3:     switch ( board_id )
4:     {
5:         case ID_IDI48: return IDI_Bank_Name_To_Symbol( board_id, name, (IDI_BANK_ENUM *) bank );
6:         case ID_IDO48: return IDO_Bank_Name_To_Symbol( board_id, name, (IDO_BANK_ENUM *) bank );
7:     }
8:     return -EC_BOARD;
9: }
```

3.2.1.17 Bank_Symbol_To_Name Function

C++

```
static const char * Bank_Symbol_To_Name(int board_id, int bank);
```

File

idi.c (see page 373)

Description

This is function Bank_Symbol_To_Name.

Body Source

```

1: static const char * Bank_Symbol_To_Name( int board_id, int bank )
2: {
3:     switch ( board_id )
4:     {
5:         case ID_IDI48: return IDI_Bank_Symbol_To_Name( ((struct idi_bank_info *) board_definition[board_id].bank_info), (IDI_BANK_ENUM) bank );
6:         case ID_IDO48: return IDO_Bank_Symbol_To_Name( ((struct ido_bank_info *) board_definition[board_id].bank_info), (IDO_BANK_ENUM) bank );
7:     }
8:     return NULL;
9: }
```

3.2.1.18 Bit_String_Index_By_Name Function

C++

```
int Bit_String_Index_By_Name(struct bit_string_info * table, char * name);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: int Bit_String_Index_By_Name( struct bit_string_info * table, char * name )
2: {
3:     int index;
4:     index = 0;
5:     while ( NULL != table[index].name )
6:     {
7:         if ( 0 == strcasecmp( table[index].name, name ) ) return index;
8:         index++;
9:     }
10:    return -EC_NOT_FOUND;
11: }
```

3.2.1.19 Bit_String_Report Function

C++

```
int Bit_String_Report(struct bit_string_info * table, size_t address, uint8_t value, FILE * out);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: int Bit_String_Report( struct bit_string_info * table, size_t address, uint8_t value, FILE
* out )
2: {
3:     size_t index;
4:     BOOL not_done;
5:     BOOL initial;
6:     uint8_t bit_string_value;
7:
8:     initial = true;
9:     not_done = true;
10:    index = 0;
11:    while ( not_done )
12:    {
13:        if ( NULL == table[index].name )
14:        {
15:            not_done = false;
16:        }
17:        else if ( address == table[index].register_offset )
18:        {
19:            size_t i;
20:            uint8_t mask = 0;
21:            for ( i = 0; i < table[index].bit_width; i++ ) mask = ( mask << 1 ) | 0x01;
22:            bit_string_value = value >> table[index].bit_offset;
23:            bit_string_value = bit_string_value & mask;
24:            if ( initial )
25:            {
26:                fprintf( out, ":" );
27:                initial = false;
28:            }
29:        }
30:    }
31: }
```

```

28:     fprintf( out, " %s=%d", table[index].name, (int) bit_string_value );
29: }
30: index++;
31: }
32: return SUCCESS;
33: }
```

3.2.1.20 Buffer_Init Function

C++

```
static void Buffer_Init();
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static void Buffer_Init(void)
2: {
3:     size_t index;
4:     buffer_index = 0;
5:     for ( index = 0; index < BUFFER_LENGTH; index++ )
6:     {
7:         buffer[BUFFER_COLUMN_VALUE][index] = 0;
8:         buffer[BUFFER_COLUMN_TAG][index] = 0;
9:     }
10: }
```

3.2.1.21 Buffer_Length Function

C++

```
static int Buffer_Length(size_t * length);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int Buffer_Length( size_t * length )
2: {
3:     *length = buffer_index;
4:     return SUCCESS;
5: }
```

3.2.1.22 Buffer_Maximum_Int32 Function

C++

```
static int Buffer_Maximum_Int32(int32_t (* buffer)[BUFFER_LENGTH], size_t length, double * maximum);
```

File

idi.c (see page 373)

Description

brief

Body Source

```
1: static int Buffer_Maximum_Int32( int32_t (* buffer)[BUFFER_LENGTH], size_t length, double
* maximum )
2: {
3:     int32_t max;
4:     size_t index;
5:     int32_t * buf;
6:
7:     if ( 0 == length ) return -EC_PARAMETER;
8:     max = INT32_MIN;
9:     buf = buffer[BUFFER_COLUMN_VALUE];
10:    for ( index = 0; index < length; index++ )
11:    {
12:        if ( buf[index] > max ) max = buf[index];
13:    }
14:    *maximum = (double) max;
15:    return SUCCESS;
16: }
```

3.2.1.23 Buffer_Mean_Int32 Function

C++

```
static int Buffer_Mean_Int32(int32_t (* buffer)[BUFFER_LENGTH], size_t length, double * mean);
```

File

idi.c (see page 373)

Description

brief

Body Source

```
1: static int Buffer_Mean_Int32( int32_t (* buffer)[BUFFER_LENGTH], size_t length, double *
mean )
2: {
3:     size_t index;
4:     double sum;
```

```

5: int32_t * buf;
6:
7: if ( 0 == length ) return -EC_PARAMETER;
8: sum = 0.0;
9: buf = buffer[BUFFER_COLUMN_VALUE];
10: for ( index = 0; index < length; index++ )
11: {
12:     sum += (double) buf[index];
13: }
14: *mean = sum / ((double)length);
15: return SUCCESS;
16: }
```

3.2.1.24 Buffer_Minimum_Int32 Function

C++

```
static int Buffer_Minimum_Int32(int32_t (* buffer)[BUFFER_LENGTH], size_t length, double * minimum);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int Buffer_Minimum_Int32( int32_t (* buffer)[BUFFER_LENGTH], size_t length, double
* minimum )
2: {
3:     int32_t min;
4:     size_t index;
5:     int32_t * buf;
6:
7:     if ( 0 == length ) return -EC_PARAMETER;
8:     min = INT32_MAX;
9:     buf = buffer[BUFFER_COLUMN_VALUE];
10:    for ( index = 0; index < length; index++ )
11:    {
12:        if ( buf[index] < min ) min = buf[index];
13:    }
14:    *minimum = (double) min;
15:    return SUCCESS;
16: }
```

3.2.1.25 Buffer_Peak_To_Peak_Int32 Function

C++

```
static int Buffer_Peak_To_Peak_Int32(double min, double max, double * peak_to_peak);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int Buffer_Peak_To_Peak_Int32( double min, double max, double * peak_to_peak )
2: {
3:   *peak_to_peak = max - min;
4:   return SUCCESS;
5: }
```

3.2.1.26 Buffer_Save Function

C++

```
static int Buffer_Save(FILE * out);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int Buffer_Save( FILE * out )
2: {
3:   size_t index;
4:   size_t column;
5:
6:   for ( index = 0; index < buffer_index; index++ )
7:   {
8:     for ( column = 0; column < BUFFER_COLUMN; column++ )
9:     {
10: #if defined( __MSDOS__ )
11:       fprintf( out, "%d", buffer[column][index] );
12: #else
13:       fprintf( out, "%d", buffer[column][index] );
14: #endif
15:       if ( column < (BUFFER_COLUMN - 1) ) fprintf( out, "," );
16:       else                               fprintf( out, "\n" );
17:     }
18:   }
19:   return SUCCESS;
20: }
```

3.2.1.27 Buffer_Save_Binary_Int32 Function

C++

```
static int Buffer_Save_Binary_Int32(const char * file_name);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int Buffer_Save_Binary_Int32( const char * file_name )
2: {
3:     FILE * fd;
4:     size_t index;
5:     size_t column;
6:
7:     if ( 0 == buffer_index ) return -SUCCESS;
8:
9:     fd = fopen( file_name, "wb" );
10:    if ( NULL == fd ) return -EC_FILE_ERROR;
11:
12:    for ( index = 0; index < buffer_index; index++ )
13:    {
14:        for ( column = 0; column < BUFFER_COLUMN; column++ )
15:        {
16:            fwrite( &(buffer[column][index]),
17:                    sizeof( int32_t ),
18:                    1,
19:                    fd
20:                );
21:        }
22:    }
23: #if(0)
24:
25:    fwrite( buffer,
26:            BUFFER_COLUMN * sizeof( int32_t ),
27:            buffer_index,
28:            fd
29:        );
30: #endif
31:    fclose( fd );
32:
33:    return SUCCESS;
34: }
```

3.2.1.28 Buffer_Standard_Deviation_Int32 Function

C++

```
static int Buffer_Standard_Deviation_Int32(int32_t (* buffer)[BUFFER_LENGTH], size_t length,
double mean, double * stdev);
```

File

idi.c (see page 373)

Description

Uses Bessel's correction.

Body Source

```

1: static int Buffer_Standard_Deviation_Int32( int32_t (* buffer)[BUFFER_LENGTH], size_t
length, double mean, double * stdev )
2: {
3:   size_t index;
4:   double term;
5:   double squared_sum;
6:   int32_t * buf;
7:
8:   if ( length < 2 ) return -EC_PARAMETER;
9:
10:  squared_sum = 0.0;
11:  buf = buffer[BUFFER_COLUMN_VALUE];
12:  for ( index = 0; index < length; index++ )
13:  {
14:    term = ((double) buf[index]) - mean;
15:    squared_sum += term * term;
16:  }
17:  squared_sum = squared_sum / ((double) ( length - 1));
18:  *stdev = sqrt( squared_sum );
19:  return SUCCESS;
20: }
```

3.2.1.29 Buffer_Stuff Function

C++

```
static int Buffer_Stuff(int32_t value, int32_t tag);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int Buffer_Stuff( int32_t value, int32_t tag )
2: {
3:   if ( buffer_index >= BUFFER_LENGTH ) return -EC_BUFFER_EXCEEDED;
4:   buffer[BUFFER_COLUMN_VALUE][buffer_index] = value;
5:   buffer[BUFFER_COLUMN_TAG][buffer_index] = tag;
6:   buffer_index++;
7:   return SUCCESS;
8: }
```

3.2.1.30 Character_Get Function

C++

```
BOOL Character_Get(int * character);
```

File

idi.c (see page 373)

Returns

If true, then a valid character is available at the keyboard, otherwise false.

Description

brief Obtains a key from the keyboard in a non-blocking way. This is exerpetted from AES Universal Library/Driver.

Excerpt from AES advanced Linux library/driver. COPYRIGHT NOTICE Copyright (c) 2012 by Apex Embedded Systems.

param[out] character Character from keyboard otherwise null character.

Body Source

```

1: BOOL Character_Get( int * character )
2: {
3:     int char_temp;
4:     BOOL result = false;
5: #ifdef __BORLANDC__
6:     if ( kbhit() )
7:     {
8:         char_temp = getch();
9:         result    = true;
10:    }
11:   else
12:   {
13:       char_temp = '\0';
14:   }
15: #else
16:     char_temp = '\0';
17:     result    = false;
18: #endif
19:     if ( NULL != character ) *character = char_temp;
20:     return result;
21: }
```

3.2.1.31 CMD__AS_Calibration_Gain Function

C++

```
static int CMD__AS_Calibration_Gain(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int CMD__AS_Calibration_Gain( int board_id, int argc, char * argv[] )
2: {
```

```

3: int32_t fsc;
4: int error_code;
5:
6: if ( argc > 0 )
7: {
8:   fsc = (int32_t) strtol( argv[0], NULL, 0 );
9:   error_code = AS_Calibration_Value_Write( board_id, AS_AM2301_FSC0, fsc );
10:  if ( error_code < 0 ) return error_code;
11: }
12: else
13: {
14:   error_code = AS_Calibration_Value_Read( board_id, AS_AM2301_FSC0, &fsc );
15:   if ( error_code < 0 ) return error_code;
16: #if defined( __MSDOS__ )
17:   printf( "Full Scale Calibration (FSC) = 0x%08lx, %ld\n", fsc, fsc );
18: #else
19:   printf( "Full Scale Calibration (FSC) = 0x%08x, %d\n", fsc, fsc );
20: #endif
21: }
22: return SUCCESS;
23: }
```

3.2.1.32 CMD_AS_Calibration_Offset Function

C++

```
static int CMD_AS_Calibration_Offset(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int CMD_AS_Calibration_Offset( int board_id, int argc, char * argv[] )
2: {
3:   int error_code;
4:   int32_t ofc;
5:
6:   if ( argc > 0 )
7:   {
8:     ofc = (int32_t) strtol( argv[0], NULL, 0 );
9:     error_code = AS_Calibration_Value_Write( board_id, AS_AM2301_OF0C0, ofc );
10:   }
11: else
12:   {
13:     error_code = AS_Calibration_Value_Read( board_id, AS_AM2301_OF0C0, &ofc );
14:     if ( error_code < 0 ) return error_code;
15: #if defined( __MSDOS__ )
16:   printf( "Offset Calibration (OFC) = 0x%08lx, %ld\n", ofc, ofc );
17: #else
18:   printf( "Offset Calibration (OFC) = 0x%08x, %d\n", ofc, ofc );
19: #endif
20:   }
21: return SUCCESS;
22: }
```

3.2.1.33 CMD__AS_Gancal Function

C++

```
static int CMD__AS_Gancal(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int CMD__AS_Gancal( int board_id, int argc, char * argv[ ] )
2: {
3:     uint8_t tx_buf[1] = { 0x19 };
4:     (void) argc;
5:     (void) argv;
6:
7: #define FUNCTION_NAME__ "CMD__AS_Gancal"
8: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
9:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
10: #endif
11: #undef FUNCTION_NAME__
12:
13: return AS_Opcode_Write( board_id, tx_buf, 1 );
14: }
```

3.2.1.34 CMD__AS_Ofscal Function

C++

```
static int CMD__AS_Ofscal(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int CMD__AS_Ofscal( int board_id, int argc, char * argv[ ] )
2: {
3:     uint8_t tx_buf[1] = { 0x18 };
4:     (void) argc;
5:     (void) argv;
6:
7: #define FUNCTION_NAME__ "CMD__AS_Ofscal"
8: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
9:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
```

```

10: #endif
11: #undef FUNCTION_NAME_
12:
13: return AS_Opcode_Write( board_id, tx_buf, 1 );
14:

```

3.2.1.35 CMD__AS_Parameter Function

C++

```
static int CMD__AS_Parameter(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int CMD__AS_Parameter( int board_id, int argc, char * argv[ ] )
2: {
3:     int         error_code = SUCCESS;
4:     int         argc_new;
5:     char **    argv_new;
6:     int         index;
7:     uint8_t     tx_buf[12]   = { 0x00 };
8:     uint8_t     rx_buf[12]   = { 0x00 };
9:
10:    if ( ( argc < 1 ) || ( NULL == argv ) )
11:    {
12:        argv_new = &(argv[1]);
13:        argc_new = argc - 1;
14:        printf( "Calibration offset: " );
15:        error_code = CMD__AS_Calibration_Offset( board_id, argc_new, argv_new );
16:        if ( error_code < 0 ) goto CMD__AS_PARAMETER_END;
17:        printf( "Calibration gain: " );
18:        error_code = CMD__AS_Calibration_Gain( board_id, argc_new, argv_new );
19:        if ( error_code < 0 ) goto CMD__AS_PARAMETER_END;
20:        printf( "Registers:\n" );
21:        for ( index = 0; index < AS_ADS1259_REGISTER_QTY; index++ )
22:        {
23:            error_code = CMD__AS_RReg( board_id, 1, &(as_register_name[index]) );
24:            if ( error_code < 0 ) goto CMD__AS_PARAMETER_END;
25:        }
26:
27:        printf( "Bit String Parameters:\n" );
28:        index = 0;
29:        while ( NULL != as_bit_string[index].name )
30:        {
31:            struct bit_string_info * bsd;
32:            size_t i;
33:            uint8_t mask = 0;
34:            bsd = &(as_bit_string[index]);
35:            error_code = AS_Registers_Read( board_id, bsd->register_offset, 1, tx_buf, rx_buf );
36:            if ( error_code < 0 ) goto CMD__AS_PARAMETER_END;
37:
38:            for ( i = 0; i < bsd->bit_width; i++ ) mask = ( mask << 1 ) | 0x01;
39:            rx_buf[2] = rx_buf[2] >> bsd->bit_offset;

```

```

40:     rx_buf[2] = rx_buf[2] & mask;
41:     printf( " %8s = %d\n", bsd->name, (int) rx_buf[2] );
42:
43:     index++;
44: }
45: }
46: if ( argc >= 2 )
47: {
48:     if ( 0 == strcmpi( "ofc", argv[0] ) )
49:     {
50:         argv_new = &(argv[1]);
51:         argc_new = argc - 1;
52:         error_code = CMD__AS_Calibration_Offset( board_id, argc_new, argv_new );
53:     }
54:     else if ( 0 == strcmpi( "fsc", argv[0] ) )
55:     {
56:         argv_new = &(argv[1]);
57:         argc_new = argc - 1;
58:         error_code = CMD__AS_Calibration_Gain( board_id, argc_new, argv_new );
59:     }
60:     else if ( ( index = AS_Register_Name_To_Offset( argv[0] ) ) >= 0 )
61:     {
62:         error_code = CMD__AS_Reg_Write_By_Name( board_id, argc, argv );
63:     }
64:     else if ( ( index = Bit_String_Index_By_Name( as_bit_string, argv[0] ) ) >= 0 )
65:     {
66:         struct bit_string_info * bsd;
67:         size_t temp;
68:         uint8_t mask = 0;
69:         bsd = &(as_bit_string[index]);
70:         error_code = AS_Registers_Read( board_id, bsd->register_offset, 1, tx_buf, rx_buf );
71:
72:         for ( temp = 0; temp < bsd->bit_width; temp++ ) mask = ( mask << 1 ) | 0x01;
73:         temp = (size_t) strtol( argv[1], NULL, 0 );
74:         temp = temp & mask;
75:         temp = temp << bsd->bit_offset;
76:         rx_buf[2] = rx_buf[2] & ~( mask << bsd->bit_offset );
77:         tx_buf[2] = rx_buf[2] | ( (uint8_t) temp );
78:         error_code = AS_Registers_Write( board_id, bsd->register_offset, 1, tx_buf, rx_buf );
79:     }
80:     else
81:     {
82:         error_code = -EC_NOT_FOUND;
83:     }
84: }
85: else
86: {
87:     if ( 0 == strcmpi( "ofc", argv[0] ) )
88:     {
89:         argv_new = &(argv[1]);
90:         argc_new = argc - 1;
91:         error_code = CMD__AS_Calibration_Offset( board_id, argc_new, argv_new );
92:     }
93:     else if ( 0 == strcmpi( "fsc", argv[0] ) )
94:     {
95:         argv_new = &(argv[1]);
96:         argc_new = argc - 1;
97:         error_code = CMD__AS_Calibration_Gain( board_id, argc_new, argv_new );
98:     }
99:     else if ( ( index = AS_Register_Name_To_Offset( argv[0] ) ) >= 0 )
100:    {
101:        error_code = CMD__AS_RReg( board_id, argc, argv );
102:    }
103:    else if ( ( index = Bit_String_Index_By_Name( as_bit_string, argv[0] ) ) >= 0 )

```

```

104: {
105:     struct bit_string_info * bsd;
106:     size_t i;
107:     uint8_t mask = 0;
108:     bsd = &(as_bit_string[index]);
109:     error_code = AS_Registers_Read( board_id, bsd->register_offset, 1, tx_buf, rx_buf );
110:     if ( error_code < 0 ) goto CMD__AS_PARAMETER_END;
111:
112:     for ( i = 0; i < bsd->bit_width; i++ ) mask = ( mask << 1 ) | 0x01;
113:     rx_buf[2] = rx_buf[2] >> bsd->bit_offset;
114:     rx_buf[2] = rx_buf[2] & mask;
115:     printf( "%8s = %d\n", bsd->name, (int) rx_buf[2] );
116: }
117: else
118: {
119:     error_code = -EC_NOT_FOUND;
120: }
121: }
122: CMD__AS_PARAMETER_END:
123: return error_code;
124: }
```

3.2.1.36 CMD__AS_Rdata Function

C++

```
static int CMD__AS_Rdata(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Description

brief In order to use Data Read in Stop Continuous Mode one must issue the following command sequence: b idi spi mode 1 b idi spi clk 100000 b idi as sdatac b idi as wreg 0 0x85 0x90 0x00 0x00 0x00 0x00 0x00 0x40 b idi as rreg 0 9 b idi as start b idi as rdata 1024 --file out.txt

Body Source

```

1: static int CMD__AS_Rdata( int board_id, int argc, char * argv[] )
2: {
3:     char baton[4] = { '/', '-', '\\', '|' };
4:     const size_t baton_max = 3;
5:     size_t baton_index;
6:     int32_t adc_value;
7:
8:
9:     size_t index;
10:    size_t loop_count_per_data_ready;
11:    size_t sample_qty;
12:    FILE * fd_text;
13:    char * file_name_text;
14:    char * file_name_binary;
15:    size_t file_request;
16:    size_t file_valid;
17:    BOOL verbose;
18:    BOOL use_environment_ads1259_registers;
19:    int error_code;
20:    struct board_dataset * dataset;
```

```

21: const size_t FILE_TYPE_TEXT = 0x01;
22: const size_t FILE_TYPE_BINARY = 0x02;
23:
24:
25:
26: STOPWATCH_HANDLE_TYPE stopwatch_handle[STOPWATCH_QUANTITY];
27: enum { TIME_OVERALL = 0, TIME_READING = 1, TIME_WRITING = 2 };
28:
29: (void) argc;
30: (void) argv;
31:
32: dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
33:
34: #define FUNCTION_NAME__ "CMD_AS_Rdata"
35: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
36: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
37: #endif
38: #undef FUNCTION_NAME__
39:
40: fd_text = NULL;
41: file_request = 0;
42: file_valid = 0;
43: index = 0;
44: sample_qty = 1;
45: verbose = false;
46: use_environment_ads1259_registers = false;
47:
48: while ( index < argc )
49: {
50:   if ( ( FILE_TYPE_TEXT | FILE_TYPE_BINARY ) & file_request )
51:   {
52:     if ( FILE_TYPE_TEXT & file_request )
53:     {
54:       file_name_text = argv[index];
55:       file_request = file_request & ~FILE_TYPE_TEXT;
56:       file_valid = file_valid | FILE_TYPE_TEXT;
57:
58:       fd_text = fopen( file_name_text, "wt" );
59:     }
60:     if ( FILE_TYPE_BINARY & file_request )
61:     {
62:       file_name_binary = argv[index];
63:       file_request = file_request & ~FILE_TYPE_BINARY;
64:       file_valid = file_valid | FILE_TYPE_BINARY;
65:     }
66:   }
67:   else if ( 0 == strcasecmp( "--text", argv[index] ) )
68:   {
69:     file_request = file_request | FILE_TYPE_TEXT;
70:   }
71:   else if ( 0 == strcasecmp( "--binary", argv[index] ) )
72:   {
73:     file_request = file_request | FILE_TYPE_BINARY;
74:   }
75:   else if ( 0 == strcasecmp( "--v", argv[index] ) )
76:   {
77:     verbose = true;
78:   }
79:   else if ( 0 == strcasecmp( "--e", argv[index] ) )
80:   {
81:     use_environment_ads1259_registers = true;
82:   }
83:   else
84:   {

```

```
85:     sample_qty = (size_t) strtol( argv[index], NULL, 0 );
86:     if ( sample_qty > ( BUFFER_LENGTH - 1 ) ) sample_qty = BUFFER_LENGTH;
87: }
88: index++;
89: }
90:
91: CMD__AS_Reset( board_id, 0, NULL );
92: Buffer_Init();
93:
94:
95: StopWatch_Initialization();
96:
97: stopwatch_handle[TIME_OVERALL] = StopWatch_Open();
98: StopWatch_Set( stopwatch_handle[TIME_OVERALL], STOPWATCH_RESET );
99: stopwatch_handle[TIME_READING] = StopWatch_Open();
100: StopWatch_Set( stopwatch_handle[TIME_READING], STOPWATCH_RESET );
101: stopwatch_handle[TIME_WRITING] = StopWatch_Open();
102: StopWatch_Set( stopwatch_handle[TIME_WRITING], STOPWATCH_RESET );
103:
104:
105:
106: #if defined( __MSDOS__ )
107: {
108:     double scratch;
109:
110:     scratch = ADS1259_CLOCK_MHZ * ( 1.0 + ADS1259_CLOCK_TOLERANCE );
111:     scratch = 65536.0 / scratch;
112:     scratch = scratch * 1000.0;
113:     scratch = scratch + 100.0;
114:     delay ( (size_t) scratch );
115: }
116: #endif
117: CMD__AS_Sdatac( board_id, 0, NULL );
118: CMD__AS_Stop( board_id, 0, NULL );
119:
120:
121:
122: loop_count_per_data_ready = 0;
123:
124:
125: if ( use_environment_ads1259_registers )
126: {
127:     uint8_t tx_buf[12] = { 0x00 };
128:     uint8_t rx_buf[12] = { 0x00 };
129:
130:     for ( index = 0; index < AS_ADS1259_REGISTER_QTY; index ++ )
131:     {
132:         tx_buf[index + 2] = dataset->ads1259.reg[index];
133:     }
134:
135:     AS_Registers_Write( board_id, AS_ADS1259_CONFIG0, AS_ADS1259_REGISTER_QTY, tx_buf,
136: rx_buf );
137: }
138: if ( true == verbose )
139: {
140:     CMD__AS_RReg( board_id, 0, NULL );
141:     baton_index = 0;
142:     printf( "\nActive: " );
143: }
144:
145: StopWatch_Set( stopwatch_handle[TIME_OVERALL], STOPWATCH_START );
146:
147:
```

```
148:  
149:  
150:  
151:     error_code = AS_Data_Read( board_id, &adc_value );  
152:  
153:  
154:     for ( index = 0; index < sample_qty; index++ )  
155:     {  
156:  
157:  
158:         StopWatch_Set( stopwatch_handle[TIME_READING], STOPWATCH_START );  
159:  
160:         CMD__AS_Start( board_id, 0, NULL );  
161:         loop_count_per_data_ready++;  
162:         AS_Data_Ready_Wait( board_id,  
163:             1,  
164:             &loop_count_per_data_ready  
165:         );  
166:  
167:         error_code = AS_Data_Read( board_id, &adc_value );  
168:         if ( error_code < 0 ) goto CMD__AS_Rdata_Error_Exit;  
169:  
170:  
171:         StopWatch_Set( stopwatch_handle[TIME_READING], STOPWATCH_PAUSE );  
172:         StopWatch_Set( stopwatch_handle[TIME_WRITING], STOPWATCH_START );  
173:  
174:         Buffer_Stuff( adc_value, loop_count_per_data_ready );  
175:  
176:         StopWatch_Set( stopwatch_handle[TIME_WRITING], STOPWATCH_PAUSE );  
177:  
178:         if ( true == verbose )  
179:         {  
180:             printf( "\b%c", baton[baton_index] );  
181:             if ( baton_index == baton_max ) baton_index = 0;  
182:             else  
183:                 baton_index++;  
184:         }  
185:  
186:         StopWatch_Set( stopwatch_handle[TIME_OVERALL], STOPWATCH_PAUSE );  
187:  
188:         if ( true == verbose )  
189:         {  
190:             printf( "\n" );  
191:         }  
192:  
193:         if ( FILE_TYPE_TEXT & file_valid )  
194:         {  
195:             Buffer_Save( fd_text );  
196:             AS_Rdata_Statistics( fd_text );  
197:             AS_Time_Statistics( stopwatch_handle, TIME_OVERALL, TIME_READING, TIME_WRITING, fd_text );  
198:             fclose( fd_text );  
199:         }  
200:         if ( FILE_TYPE_BINARY & file_valid )  
201:         {  
202:             Buffer_Save_Binary_Int32( file_name_binary );  
203:         }  
204:  
205:  
206:         AS_Time_Statistics( stopwatch_handle, TIME_OVERALL, TIME_READING, TIME_WRITING, stdout );  
207:  
208:  
209:         AS_Rdata_Statistics( stdout );  
210:
```

```

211: CMD__AS_Stop( board_id, 0, NULL );
212:
213: StopWatch_Termination();
214:
215: return SUCCESS;
216: CMD__AS_Rdatac_Error_Exit:
217: AS_Chip_Select_Route_Restore( board_id );
218: if ( FILE_TYPE_TEXT & file_valid ) fclose( fd_text );
219:
220: StopWatch_Termination();
221:
222: return error_code;
223: }
```

3.2.1.37 CMD__AS_Rdatac Function

C++

```
static int CMD__AS_Rdatac(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int CMD__AS_Rdatac( int board_id, int argc, char * argv[] )
2: {
3:     uint8_t tx_buf[1] = { 0x10 };
4:     (void) argc;
5:     (void) argv;
6:
7: #define FUNCTION_NAME__ "CMD__AS_Rdatac"
8: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
9:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
10: #endif
11: #undef FUNCTION_NAME__
12:
13: return AS_Opcode_Write( board_id, tx_buf, 1 );
14: }
```

3.2.1.38 CMD__AS_Reg_Write_By_Name Function

C++

```
static int CMD__AS_Reg_Write_By_Name(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int CMD__AS_Reg_Write_By_Name( int board_id, int argc, char * argv[] )
2: {
3:     size_t count;
4:     size_t index;
5:     size_t address;
6:     size_t value;
7:     int error_code;
8:     uint8_t tx_buf[12] = { 0x00 };
9:     uint8_t rx_buf[12] = { 0x00 };
10:
11: #define FUNCTION_NAME__ "CMD__AS_Reg_Write_By_Name"
12: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
13:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
14: #endif
15: #undef FUNCTION_NAME__
16:
17: if ( argc < 2 ) return -EC_SYNTAX;
18:
19: index = 0;
20: while ( ( argc - index ) > 1 )
21: {
22:     if ( ( argc - index ) > 1 )
23:     {
24:         if ( ( argv[index][0] >= '0' ) && ( argv[index][0] <= '9' ) )
25:         {
26:             address = (size_t) strtol( argv[index], NULL, 0 );
27:         }
28:     else
29:     {
30:         error_code = AS_Register_Name_To_Offset( argv[index] );
31:         if ( error_code < 0 ) return -EC_NOT_FOUND;
32:         address = (size_t) error_code;
33:     }
34:     index++;
35:
36:     if ( ( argc - index ) > 0 )
37:     {
38:         value = (size_t) strtol( argv[index], NULL, 0 );
39:         tx_buf[2] = (uint8_t) value;
40:         count = 1;
41:         error_code = AS_Registers_Write( board_id, address, count, tx_buf, rx_buf );
42:     }
43:     else
44:     {
45:         return -EC_SYNTAX;
46:     }
47:     index++;
48: }
49: else
50: {
51:     error_code = -EC_SYNTAX;
52: }
53: }
54:
55: if ( (argc - index) > 0 ) error_code = -EC_EXTRA_IGNORED;
56: return error_code;
57: }
```

3.2.1.39 CMD__AS_Register_Load Function

C++

```
static int CMD__AS_Register_Load(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Description

brief

Body Source

```
1: static int CMD__AS_Register_Load( int board_id, int argc, char * argv[] )
2: {
3:     int         error_code;
4:     FILE *      fd;
5:     void *      data_pointer = NULL;
6:     uint8_t     tx_buf[12]   = { 0x00 };
7:     uint8_t     rx_buf[12]   = { 0x00 };
8:     struct board_dataset * dataset;
9:
10:    dataset     = ( struct board_dataset * ) board_definition[board_id].dataset;
11:    error_code = SUCCESS;
12:
13:    if ( 0 == argc )
14:    {
15:        size_t index;
16:        for ( index = 0; index < AS_AM2315_REGISTER_QTY; index ++ )
17:        {
18:            tx_buf[index + 2] = dataset->ads1259.reg[index];
19:        }
20:
21:        error_code = AS_Registers_Write( board_id, AS_AM2315_CONFIG0, AS_AM2315_REGISTER_QTY,
22:                                         tx_buf, rx_buf );
23:    }
24:    else
25:    {
26:        fd = fopen( argv[0], "r" );
27:        if ( NULL == fd )
28:        {
29:            error_code = -EC_FILE_ERROR;
30:        }
31:        else
32:        {
33:            size_t file_size;
34:
35:            fseek( fd, 0L, SEEK_END );
36:            file_size = (size_t) ftell( fd );
37:            fseek( fd, 0L, SEEK_SET );
38:            if ( file_size != AS_AM2315_REGISTER_QTY )
39:            {
40:                error_code = -EC_FILE_ERROR_SIZE;
41:            }
42:        }
43:        data_pointer = (void *) &(tx_buf[2]);
```

```

44:     fread( data_pointer, AS_AM2315_REGISTER_QTY, 1, fd );
45:     error_code = AS_Registers_Write( board_id, AS_AM2315_CONFIG0,
AS_AM2315_REGISTER_QTY, tx_buf, rx_buf );
46:   }
47:   fclose( fd );
48: }
49: }
50: return error_code;
51: }
```

3.2.1.40 CMD__AS_Register_Save Function

C++

```
static int CMD__AS_Register_Save(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int CMD__AS_Register_Save( int board_id, int argc, char * argv[] )
2: {
3:   int error_code;
4:   FILE * fd;
5:   uint8_t tx_buf[12] = { 0x00 };
6:   uint8_t rx_buf[12] = { 0x00 };
7:
8:   if ( argc < 1 ) return -EC_SYNTAX;
9:
10:  error_code = AS_Registers_Read( board_id, AS_AM2315_CONFIG0, AS_AM2315_REGISTER_QTY,
tx_buf, rx_buf );
11:  if ( error_code < 0 ) return error_code;
12:
13:  fd = fopen( argv[0], "w" );
14:  if ( NULL != fd )
15:  {
16:    void * data_pointer = (void *) &(rx_buf[2]);
17:    fwrite( data_pointer, AS_AM2315_REGISTER_QTY, 1, fd );
18:    fclose( fd );
19:  }
20:  return SUCCESS;
21: }
```

3.2.1.41 CMD__AS_Reset Function

C++

```
static int CMD__AS_Reset(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int CMD__AS_Reset( int board_id, int argc, char * argv[ ] )
2: {
3:     uint8_t tx_buf[1] = { 0x07 };
4:     (void) argc;
5:     (void) argv;
6:
7: #define FUNCTION_NAME__ "CMD__AS_Reset"
8: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
9:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
10: #endif
11: #undef FUNCTION_NAME__
12:
13: return AS_Opcode_Write( board_id, tx_buf, 1 );
14: }
```

3.2.1.42 CMD__AS_RReg Function

C++

```
static int CMD__AS_RReg(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int CMD__AS_RReg( int board_id, int argc, char * argv[ ] )
2: {
3:     size_t count;
4:     size_t index;
5:     size_t address;
6:     int error_code;
7:     BOOL dump_as_hex;
8:     uint8_t tx_buf[12] = { 0x00 };
9:     uint8_t rx_buf[12] = { 0x00 };
10:
11: #define FUNCTION_NAME__ "CMD__AS_RReg"
12: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
13:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
14: #endif
15: #undef FUNCTION_NAME__
16:
17: dump_as_hex = false;
18: index = 0;
19: if ( ( argc < 1 ) || ( NULL == argv ) )
20: {
21:     count = AS_ADS1259_REGISTER_QTY;
22:     address = AS_ADS1259_CONFIG0;
```

```

23:     index = 1;
24: }
25: else
26: {
27:     if ( ( 'a' == argv[0][0] ) || ( 'A' == argv[0][0] ) )
28:     {
29:         count    = AS_AM2315_REGISTER_QTY;
30:         address = AS_AM2315_CONFIG0;
31:         index   = 1;
32:     }
33: else if ( 1 == argc )
34: {
35:     count    = 1;
36:     error_code = AS_Register_Name_To_Offset( argv[0] );
37:     if ( error_code < 0 )
38:     {
39:         address = (size_t) strtol( argv[0], NULL, 0 );
40:     }
41: else
42: {
43:     address = (size_t) error_code;
44: }
45: index   = 1;
46: }
47: else
48: {
49:
50:     count    = (size_t) strtol( argv[1], NULL, 0 );
51:     error_code = AS_Register_Name_To_Offset( argv[0] );
52:     if ( error_code < 0 )
53:     {
54:         address = (size_t) strtol( argv[0], NULL, 0 );
55:     }
56: else
57: {
58:     address = (size_t) error_code;
59: }
60: index   = 2;
61: }
62:
63: if ( argc > index )
64: {
65:     if ( ( 'h' == argv[index][0] ) || ( 'H' == argv[index][0] ) )
66:     {
67:         dump_as_hex = true;
68:     }
69: }
70: }
71:
72: error_code = AS_Registers_Read( board_id, address, count, tx_buf, rx_buf );
73: if ( error_code < 0 ) return error_code;
74:
75: if ( dump_as_hex )
76: {
77:     Hex_Dump_Line( (uint16_t) address, count, &(rx_buf[2]), stdout );
78: }
79: else
80: {
81:     for ( index = 0; index < count; index++ )
82:     {
83:         printf( "%s=0x%02X", as_register_name[address + index], rx_buf[index + 2] );
84:         Bit_String_Report( as_bit_string, address + index, rx_buf[index + 2], stdout );
85:         printf( "\n" );
86:     }
}

```

```
87: }
88: return SUCCESS;
89: }
```

3.2.1.43 CMD__AS_Sdatac Function

C++

```
static int CMD__AS_Sdatac(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Description

brief

Body Source

```
1: static int CMD__AS_Sdatac( int board_id, int argc, char * argv[] )
2: {
3:     uint8_t tx_buf[1] = { 0x11 };
4:     (void) argc;
5:     (void) argv;
6:
7: #define FUNCTION_NAME__ "CMD__AS_Sdatac"
8: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
9:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
10: #endif
11: #undef FUNCTION_NAME__
12:
13: return AS_Opcode_Write( board_id, tx_buf, 1 );
14: }
```

3.2.1.44 CMD__AS_Sleep Function

C++

```
static int CMD__AS_Sleep(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Description

brief

Body Source

```
1: static int CMD__AS_Sleep( int board_id, int argc, char * argv[] )
2: {
3:     uint8_t tx_buf[1] = { 0x05 };
4:     (void) argc;
5:     (void) argv;
6:
```

```

7: #define FUNCTION_NAME__ "CMD__AS_Sleep"
8: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
9: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
10: #endif
11: #undef FUNCTION_NAME__
12:
13: return AS_Opcode_Write( board_id, tx_buf, 1 );
14: }
```

3.2.1.45 CMD__AS_Start Function

C++

```
static int CMD__AS_Start(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int CMD__AS_Start( int board_id, int argc, char * argv[ ] )
2: {
3:     uint8_t tx_buf[1] = { 0x09 };
4:     (void) argc;
5:     (void) argv;
6:
7: #define FUNCTION_NAME__ "CMD__AS_Start"
8: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
9: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
10: #endif
11: #undef FUNCTION_NAME__
12:
13: return AS_Opcode_Write( board_id, tx_buf, 1 );
14: }
```

3.2.1.46 CMD__AS_Stop Function

C++

```
static int CMD__AS_Stop(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Description

brief

Body Source

```
1: static int CMD__AS_Stop( int board_id, int argc, char * argv[ ] )
```

```

2: {
3:     uint8_t tx_buf[1] = { 0x0B };
4:     (void) argc;
5:     (void) argv;
6:
7: #define FUNCTION_NAME__ "CMD__AS_Stop"
8: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
9:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
10: #endif
11: #undef FUNCTION_NAME__
12:
13: return AS_Opcode_Write( board_id, tx_buf, 1 );
14: }
```

3.2.1.47 CMD__AS_Wakeup Function

C++

```
static int CMD__AS_Wakeup(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int CMD__AS_Wakeup( int board_id, int argc, char * argv[] )
2: {
3:     uint8_t tx_buf[1] = { 0x03 };
4:     (void) argc;
5:     (void) argv;
6:
7: #define FUNCTION_NAME__ "CMD__AS_Wakeup"
8: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
9:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
10: #endif
11: #undef FUNCTION_NAME__
12:
13: return AS_Opcode_Write( board_id, tx_buf, 1 );
14: }
```

3.2.1.48 CMD__AS_WReg Function

C++

```
static int CMD__AS_WReg(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int CMD__AS_WReg( int board_id, int argc, char * argv[ ] )
2: {
3:   size_t count;
4:   size_t index;
5:   size_t address;
6:   int error_code;
7:   BOOL not_done;
8:   uint8_t tx_buf[12] = { 0x00 };
9:   uint8_t rx_buf[12] = { 0x00 };
10:
11: #define FUNCTION_NAME__ "CMD__AS_WReg"
12: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
13: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
14: #endif
15: #undef FUNCTION_NAME__
16:
17: if ( argc < 2 ) return -EC_SYNTAX;
18:
19: index = 0;
20: if ( ( argv[index][0] >= '0' ) && ( argv[index][0] <= '9' ) )
21: {
22:   address = (size_t) strtol( argv[index], NULL, 0 );
23: }
24: else
25: {
26:   error_code = AS_Register_Name_To_Offset( argv[index] );
27:   if ( error_code < 0 ) return -EC_NOT_FOUND;
28:   address = (size_t) error_code;
29: }
30: index++;
31:
32: count = 0;
33: not_done = true;
34: do
35: {
36:   tx_buf[count + 2] = (uint8_t) strtol( argv[index], NULL, 0 );
37:   count++;
38:   index++;
39:
40: if ( count >= AS_ADS1259_REGISTER_QTY )
41: {
42:   not_done = false;
43:   error_code = -EC_EXTRA_IGNORED;
44: }
45: if ( index >= argc ) not_done = false;
46: } while ( not_done );
47:
48: error_code = AS_Registers_Write( board_id, address, count, tx_buf, rx_buf );
49: return error_code;
50: }
```

3.2.1.49 CMD__FRAM_Dump Function

C++

```
static int CMD__FRAM_Dump(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__FRAM_Dump( int board_id, int argc, char * argv[] )
2: {
3:     uint16_t address;
4:     uint16_t length;
5:
6:     if ( argc < 1 ) return -EC_PARAMETER;
7:
8:     address = (uint16_t) strtol( argv[0], NULL, 0 );
9:     if ( address > ( FRAM_DENSITY_BYTES - 1 ) ) return -EC_FRAM_ADDRESS_RANGE;
10:
11:    if ( argc < 2 ) length = HEX_DUMP_BYTES_PER_LINE;
12:    else           length = (uint16_t) strtol( argv[1], NULL, 0 );
13:
14:    if ( ( address + length ) > FRAM_DENSITY_BYTES ) length = FRAM_DENSITY_BYTES - address;
15:
16:    return FRAM_Report( board_id, address, length, stdout );
17: }
```

3.2.1.50 CMD__FRAM_Init Function

C++

```
static int CMD__FRAM_Init(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__FRAM_Init( int board_id, int argc, char * argv[] )
2: {
3:     int      error_code;
4:
5:     size_t   tx_size_in_bytes;
```

```

6:  struct board_dataset * dataset;
7:  dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
8:
9:
10: if ( argc < 1 )
11: {
12:   error_code = FRAM_Set( board_id, 0, NULL );
13: }
14: else
15: {
16:   CMD__SPI_Data_Interpreter( 0,
17:                               argc,
18:                               argv,
19:                               FRAM_BLOCK_SIZE,
20:                               &tx_size_in_bytes,
21:                               dataset->spi.fram_block
22:                           );
23:
24: #if defined( IDI_FRAM_PRINT_DEBUG )
25:
26: {
27:   size_t display_count;
28:
29:   if ( tx_size_in_bytes > HEX_DUMP_BYTES_PER_LINE ) display_count = HEX_DUMP_BYTES_PER_LINE;
30:   else
31:     display_count = tx_size_in_bytes;
32:   Hex_Dump_Line( 0, display_count, dataset->spi.fram_block, stdout );
33: }
34: #endif
35:   error_code = FRAM_Set( board_id, tx_size_in_bytes, dataset->spi.fram_block );
36: }
37: return error_code;
38: }
```

3.2.1.51 CMD__FRAM_Load Function

C++

```
static int CMD__FRAM_Load(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__FRAM_Load( int board_id, int argc, char * argv[] )
2: {
3:   int error_code;
4:   uint16_t address;
5:
6:   FILE * out;
```

```

7:
8: if ( argc < 2 ) return -EC_PARAMETER;
9:
10: address = (uint16_t) strtol( argv[0], NULL, 0 );
11: if ( address > ( FRAM_DENSITY_BYTES - 1 ) ) return -EC_FRAM_ADDRESS_RANGE;
12: out = fopen( argv[2], "r" );
13: error_code = FRAM_File_To_Memory( board_id, address, 0 , out );
14: fclose( out );
15: return error_code;
16: }
```

3.2.1.52 CMD__FRAM_RDID Function

C++

```
static int CMD__FRAM_RDID(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__FRAM_RDID( int board_id, int argc, char * argv[] )
2: {
3:     uint32_t id;
4:     (void) argc;
5:     (void) argv;
6:     FRAM__Read_ID( board_id, &id );
7: #if defined( __MSDOS__ )
8:     printf( "FRAM ID: 0x%08lX\n", id );
9: #else
10:    printf( "FRAM ID: 0x%08X\n", id );
11: #endif
12:    return SUCCESS;
13: }
```

3.2.1.53 CMD__FRAM_RDSR Function

C++

```
static int CMD__FRAM_RDSR(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__FRAM_RDSR( int board_id, int argc, char * argv[] )
2: {
3:     int     error_code;
4:     uint8_t   status;
5:     (void)    argc;
6:     (void)    argv;
7:     error_code = FRAM__Read_Status_Register( board_id, &status );
8:     printf( "FRAM STATUS: 0x%02X\n", ((int) status) );
9:     return error_code;
10: }
```

3.2.1.54 CMD__FRAM_Save Function

C++

```
static int CMD__FRAM_Save(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__FRAM_Save( int board_id, int argc, char * argv[] )
2: {
3:     int     error_code;
4:     uint16_t   address;
5:     size_t    length;
6:     FILE *      out;
7:
8:     if ( argc < 3 ) return -EC_PARAMETER;
9:
10:    address = (uint16_t) strtol( argv[0], NULL, 0 );
11:    if ( address > ( FRAM_DENSITY_BYTES - 1 ) ) return -EC_FRAM_ADDRESS_RANGE;
12:
13:    length   = (uint16_t) strtol( argv[1], NULL, 0 );
14:    if ( (address + length - 1) > ( FRAM_DENSITY_BYTES - 1 ) ) return -EC_FRAM_ADDRESS_RANGE;
15:    out = fopen( argv[2], "w" );
16:    error_code = FRAM_Memory_To_File( board_id, address, length, out );
17:    fclose( out );
```

```
18:     return error_code;
19: }
```

3.2.1.55 CMD__FRAM_WRDI Function

C++

```
static int CMD__FRAM_WRDI(int board_id, int argc, char * argv[ ]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```
1: static int CMD__FRAM_WRDI( int board_id, int argc, char * argv[ ] )
2: {
3:     (void)    argc;
4:     (void)    argv;
5:     return FRAM__Write_Disable( board_id );
6: }
```

3.2.1.56 CMD__FRAM_WREN Function

C++

```
static int CMD__FRAM_WREN(int board_id, int argc, char * argv[ ]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```
1: static int CMD__FRAM_WREN( int board_id, int argc, char * argv[ ] )
2: {
3:     (void)    argc;
4:     (void)    argv;
5:     return FRAM__Write_Enable_Latch_Set( board_id );
6: }
```

3.2.1.57 CMD__FRAM_Write Function

C++

```
static int CMD__FRAM_Write(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__FRAM_Write( int board_id, int argc, char * argv[] )
2: {
3:     int     error_code;
4:     size_t   count;
5:     uint16_t address;
6:     struct board_dataset * dataset;
7:     dataset    = ( struct board_dataset * ) board_definition[board_id].dataset;
8:
9:     if ( argc < 2 ) return -EC_PARAMETER;
10:
11:    address = (uint16_t) strtol( argv[0], NULL, 0 );
12:    if ( address > ( FRAM_DENSITY_BYTES - 1 ) ) return -EC_FRAM_ADDRESS_RANGE;
13:
14:    error_code = CMD__SPI_Data_Interpreter( 1,
15:                                            argc,
16:                                            argv,
17:                                            FRAM_BLOCK_SIZE,
18:                                            &count,
19:                                            dataset->spi.fram_block
20:                                         );
21:    if ( SUCCESS != error_code ) goto CMD__FRAM_Write_Error_Exit;
22:    error_code = FRAM_Memory_Write( board_id, address, count, dataset->spi.fram_block );
23:    CMD__FRAM_Write_Error_Exit:
24:        return error_code;
25: }
```

3.2.1.58 CMD__FRAM_WRSR Function

C++

```
static int CMD__FRAM_WRSR(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__FRAM_WRSR( int board_id, int argc, char * argv[ ] )
2: {
3:
4:     uint8_t status;
5:
6:     if ( argc < 1 ) return -EC_PARAMETER;
7:     status = (uint8_t) strtol( argv[0], NULL, 0 );
8:     return FRAM__Write_Status_Register( board_id, status );
9: }
```

3.2.1.59 CMD__IAI16_AI_All Function

C++

```
static int CMD__IAI16_AI_All(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__IAI16_AI_All( int board_id, int argc, char * argv[ ] )
2: {
3:     int error_code;
4:     int channel;
5:     int value;
6:
7:     if ( argc < 1 ) return -EC_NOT_FOUND;
8:
9:     channel = (int) strtol( argv[0], NULL, 0 );
10:
11:
12:     IAI16_AI_Channel_Multiple_Get( board_id,
13:                                     0,
14:                                     IAI16_CHANNEL_QTY - 1,
15:                                     0,
16:                                     table
17:                                     );
18: }
```

```

19: for ( channel = 0; channel < IAI16_CHANNEL_QTY; channel++ )
20: {
21:     printf( "IA%02d: 0x%04X %d\n", channel, table[channel], table[channel] );
22: }
23:
24:
25:
26:
27:
28:
29:
30:
31: return SUCCESS;
32: }
```

3.2.1.60 CMD__IAI16_AI_Channel Function

C++

```
static int CMD__IAI16_AI_Channel(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__IAI16_AI_Channel( int board_id, int argc, char * argv[] )
2: {
3:     int error_code;
4:     int channel;
5:     int value;
6:
7:     if ( argc < 1 ) return -EC_NOT_FOUND;
8:
9:     channel = (int) strtol( argv[0], NULL, 0 );
10:
11:    if ( argc < 2 )
12:    {
13:        error_code = IAI_AI_Channel_Get( board_id, channel, &value );
14:        printf( "IA%02d: 0x%04X %d\n", channel, value, value );
15:    }
16:    return SUCCESS;
17: }
```

3.2.1.61 CMD__IAI16_AI_ID Function

C++

```
static int CMD__IAI16_AI_ID(int board_id, int argc, char * argv[]);
```

Copyright ©2015 by [Apex Embedded Systems](#). All rights reserved.

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```
1: static int CMD__IAI16_AI_ID( int board_id, int argc, char * argv[] )  
2: {  
3:     uint16_t id;  
4:     (void) argc;  
5:     (void) argv;  
6:     IAI_AI_ID_Get( board_id, &id );  
7:     printf( "IAI16 AI ID: 0x%04X\n", id );  
8:     return SUCCESS;  
9: }
```

3.2.1.62 CMD__IDI48_DIN_All Function

C++

```
static int CMD__IDI48_DIN_All(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```
1: static int CMD__IDI48_DIN_All( int board_id, int argc, char * argv[] )  
2: {  
3:     int error_code;  
4:     int channel;  
5:  
6:     int cp;  
7:     enum { MODE_NONE = 0, MODE_BINARY = 1, MODE_HEX = 2, MODE_ALL = 3 } mode_out;  
8:     int group;  
9:     char message[64];  
10:    uint8_t din_grp[IDI_DIN_GROUP_QTY];  
11:    uint8_t mask;  
12:    (void) argc;  
13:    (void) argv;
```

```

14:
15: mode_out = MODE_ALL;
16: if ( argc > 0 )
17: {
18:     int index;
19:     mode_out = MODE_NONE;
20:     for ( index = 0; index < argc; index++ )
21:     {
22:         if ( 0 == strcasecmp( "binary", argv[index] ) ) mode_out |= MODE_BINARY;
23:         else if ( 0 == strcasecmp( "group", argv[index] ) ) mode_out |= MODE_HEX;
24:         else if ( 0 == strcasecmp( "hex", argv[index] ) ) mode_out |= MODE_HEX;
25:         else mode_out |= MODE_ALL;
26:     }
27: }
28:
29: cp = 0;
30: group = 0;
31: for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
32: {
33:     error_code = IDI_DIN_Group_Get( board_id, group, &(din_grp[group]) );
34:     mask = 0x01;
35:     for ( channel = 0; channel < 8; channel++ )
36:     {
37:         message[cp++] = !(din_grp[group] & mask) ? '1' : '0';
38:         mask = mask << 1;
39:     }
40:     message[cp++] = ' ';
41: }
42: message[cp] = '\0';
43: if ( MODE_BINARY == ( mode_out & MODE_BINARY ) )
44: {
45:     printf( "DIN: %s\n", message );
46: }
47: if ( MODE_HEX == ( mode_out & MODE_HEX ) )
48: {
49:     printf( "DIN:" );
50:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ ) printf( " %02X", din_grp[group] );
51:     printf( "\n" );
52: }
53:
54: return SUCCESS;
55: }
```

3.2.1.63 CMD_IDI48_DIN_Channel Function

C++

```
static int CMD_IDI48_DIN_Channel(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__IDI48_DIN_Channel( int board_id, int argc, char * argv[] )
2: {
3:     int error_code;
4:     int channel;
5:     char message[IDI_DIN_GROUP_QTY + 2];
6:     BOOL value;
7:
8:     if ( argc < 1 ) return -EC_NOT_FOUND;
9:
10:    channel = (int) strtol( argv[0], NULL, 0 );
11:    error_code = IDI_DIN_Channel_Get( board_id, channel, &value );
12:    message[0] = value ? '1' : '0';
13:    message[1] = '\0';
14:
15:    printf( "DIN%02d: %s\n", channel, message );
16:    return SUCCESS;
17: }
```

3.2.1.64 CMD__IDI48_DIN_Group Function

C++

```
static int CMD__IDI48_DIN_Group(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__IDI48_DIN_Group( int board_id, int argc, char * argv[] )
2: {
3:     int error_code;
4:
5:     int group;
6:     int group_count;
7:     BOOL do_all;
8:     uint8_t din_grp[IDI_DIN_GROUP_QTY];
9:
10:    if ( argc < 1 ) do_all = true;
11:    else
12:        do_all = false;
13:
14:    if ( 0 == strcmpi( "all", argv[0] ) ) do_all = true;
15:
16:    if ( do_all )
17:    {
18:        for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
19:            error_code = IDI_DIN_Group_Get( board_id, group, &(din_grp[group]) );
```

```

20:    }
21:    group_count = IDI_DIN_GROUP_QTY;
22:  }
23: else
24: {
25:   group = (int) strtol( argv[0], NULL, 0 );
26:   error_code = IDI_DIN_Group_Get( board_id, group, &(din_grp[0]) );
27:   group_count = 1;
28: }
29:
30: printf( "DIN_GROUP:" );
31: for ( group = 0; group < group_count; group++ )
32: {
33:   printf( " 0x%02X", ((int) din_grp[group]) );
34: }
35: printf( "\n" );
36: return SUCCESS;
37: }
```

3.2.1.65 CMD_IDI48_DIN_Helper_All_Values_False Function

C++

```
static BOOL CMD_IDI48_DIN_Helper_All_Values_False(uint8_t * dg, size_t group_qty);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static BOOL CMD_IDI48_DIN_Helper_All_Values_False( uint8_t * dg, size_t group_qty )
2: {
3:   size_t group;
4:
5:   for ( group = 0; group < group_qty; group++ )
6:   {
7:     if ( 0x00 != dg[group] ) return false;
8:   }
9:   return true;
10: }
```

3.2.1.66 CMD_IDI48_DIN_ID Function

C++

```
static int CMD_IDI48_DIN_ID(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__IDI48_DIN_ID( int board_id, int argc, char * argv[ ] )
2: {
3:     uint16_t id;
4:     (void) argc;
5:     (void) argv;
6:
7:     if ( argc > 0 )
8:     {
9:         if ( 0 == strcmpi( "scan", argv[0] ) )
10:        {
11:            size_t row;
12:            int port;
13:            int error_code = SUCCESS;
14:
15:            printf ( "Scanning: " );
16:            error_code = IDI48_DIN_ID_Port_Scan( lpm_lx800_port_list, &row, &port );
17:            if ( SUCCESS == error_code )
18:            {
19:                printf( "port found at row = %02d, id address = 0x%04X", (int) row, port );
20:            }
21:            else
22:            {
23:                printf( "not found" );
24:            }
25:            printf( "\n" );
26:
27:        }
28:    }
29:    else
30:    {
31:        IDI_DIN_ID_Get( board_id, &id );
32:        printf( "IDI48 DIN ID: 0x%04X\n", id );
33:    }
34:    return SUCCESS;
35: }
```

3.2.1.67 CMD__IDI48_DIN_Test Function

C++

```
static int CMD__IDI48_DIN_Test(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__IDI48_DIN_Test( int board_id, int argc, char * argv[] )
2: {
3:     int     error_code;
4:     int     index;
5:     int     argc_new;
6:     char ** argv_new;
7:     char *     endptr;
8:
9:
10:    if ( argc < 1 ) return -EC_NOT_FOUND;
11:
12:    error_code = -EC_SYNTAX;
13:
14:
15:    strtol( argv[0], &endptr, 0 );
16:    if ( argv[0] != endptr )
17:    {
18:        error_code = (* cmd_idi48_test[0].cmd_fnc )( board_id, argc, argv );
19:    }
20:    else
21:    {
22:        index = 0;
23:        while ( NULL != cmd_idi48_test[index].cmd_fnc )
24:        {
25:            if ( 0 == strcasecmp( cmd_idi48_test[index].name, argv[0] ) )
26:            {
27:                argv_new = &(argv[1]);
28:                argc_new = argc - 1;
29:                if ( 0 == argc_new ) argv_new = NULL;
30:                error_code = (* cmd_idi48_test[index].cmd_fnc )( board_id, argc_new, argv_new );
31:                break;
32:            }
33:            index++;
34:        }
35:    }
36:    return error_code;
37: }
```

3.2.1.68 CMD__IDI48_DIN_Test_FE Function

C++

```
static int CMD__IDI48_DIN_Test_FE(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD_IDI48_DIN_Test_FE( int board_id, int argc, char * argv[] )
2: {
3:     int     error_code;
4:     TEST_STATE_ENUM state;
5:     size_t   group;
6:     size_t   bit;
7:     size_t   index;
8:     uint8_t   result;
9:     uint8_t   dig[IDI_DIN_GROUP_QTY];
10:    uint8_t   pending[IDI_DIN_GROUP_QTY];
11:    uint8_t   mask;
12:    BOOL    din_cmplt[IDI_DIN_QTY];
13: #if defined( DIN_TEST_DEBUG_PRINT )
14:     char           message[64];
15:     int      cp;
16:     int      state_detect;
17: #endif
18:     (void)   argc;
19:     (void)   argv;
20:
21:     state = TEST_STATE_INIT;
22:
23:     for( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
24:     {
25:         IDI_DIN_Group_Get( board_id, group, &(dig[group]) );
26:         IO_Write_U8( board_id, __LINE__, IDI_EDGE_GROUP0 + group, 0x00 );
27:         IO_Write_U8( board_id, __LINE__, IDI_CLEAR_GROUP0 + group, 0xFF );
28:         IO_Read_U8( board_id, __LINE__, IDI_PEND_GROUP0 + group, &(pending[group]) );
29:     }
30:     for ( bit = 0; bit < IDI_DIN_QTY; bit++ ) din_cmplt[bit] = false;
31:
32:     printf( "DIN_TEST_FALLING_EDGE:" );
33:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
34:     {
35:         IO_Read_U8( board_id, __LINE__, IDI_PEND_GROUP0 + group, &(pending[group]) );
36:         IDI_DIN_Group_Get( board_id, group, &(dig[group]) );
37:         IO_Write_U8( board_id, __LINE__, IDI_INTR_BIT_GROUP0 + group, 0xFF );
38:     }
39:     while ( ( TEST_STATE_QUIT != state ) && ( TEST_STATE_COMPLETE != state ) )
40:     {
41:         switch( state )
42:         {
43:             case TEST_STATE_INIT:
44:                 if ( true == CMD_IDI48_DIN_Helper_All_Values_False( dig, IDI_DIN_GROUP_QTY ) )
45:                 {
46:                     state = TEST_STATE_READY;
47:                     printf( " READY" );
48:                 }
49:                 for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
50:                 {
51:                     IDI_DIN_Group_Get( board_id, group, &(dig[group]) );
52:                     IO_Write_U8( board_id, __LINE__, IDI_CLEAR_GROUP0 + group, 0xFF );

```

```

53:     }
54:     break;
55: case TEST_STATE_READY:
56: #if defined( DIN_TEST_DEBUG_PRINT )
57:     cp = 0;
58:     strcpy( message, "" );
59:     state_detect = 2;
60:     printf( "\nPENDING:" );
61:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
62:     {
63:         printf( " 0x%02X", pending[group] );
64:     }
65: #else
66:     printf( "  " );
67: #endif
68:     state = TEST_STATE_ACTIVE;
69:     break;
70: case TEST_STATE_ACTIVE:
71:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
72:     {
73:         if ( 0x00 != pending[group] )
74:         {
75:             mask = 0x01;
76:             for ( bit = 0; bit < IDI_DIN_GROUP_SIZE; bit++ )
77:             {
78:                 index = bit + group * IDI_DIN_GROUP_SIZE;
79:                 if ( ( 0x00 != (mask & pending[group]) ) && ( 0x00 == (dig[group] & mask) ) )
80:                 {
81:                     if ( false == din_cmplt[index] )
82:                     {
83:                         din_cmplt[index] = true;
84:                     }
85:                 }
86:             cp += sprintf( &(message[cp]), " [%2d]", (int) index );
87:             state_detect = 1;
88:         #else
89:             printf( "." );
90:         #endif
91:         }
92:         mask = mask << 1;
93:     }
94:     IO_Write_U8( board_id, __LINE__, IDI_CLEAR_GROUP0 + group, 0xFF );
95: }
96: }
97: }
98: #if defined( DIN_TEST_DEBUG_PRINT )
99: if ( state_detect < 3 )
100: {
101:     printf( "\nPENDING:" );
102:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
103:     {
104:         printf( " 0x%02X", pending[group] );
105:     }
106:     printf( "  " );
107:     printf( "%s", message );
108:     cp = 0;
109:     state_detect++;
110: }
111: strcpy( message, "" );
112: #endif
113:     result = true;
114:     for ( bit = 0; bit < IDI_DIN_QTY; bit++ ) result = result & din_cmplt[bit];
115:     if ( true == result ) state = TEST_STATE_COMPLETE;
116:

```

```

117:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
118:     {
119:         IO_Read_U8( board_id, __LINE__, IDI_PEND_GROUP0 + group, &(pending[group]) );
120:         IDI_DIN_Group_Get( board_id, group, &(dig[group]) );
121:     }
122:     break;
123: case TEST_STATE_COMPLETE:
124: #if defined( DIN_TEST_DEBUG_PRINT )
125:     strcpy( message, "" );
126:     printf( "\nPENDING:" );
127:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
128:     {
129:         printf( " 0x%02X", pending[group] );
130:     }
131:     printf( "  " );
132:     printf( "\n" );
133: #endif
134:     break;
135: case TEST_STATE_QUIT:
136:     break;
137: default:
138:     break;
139: }
140: if ( true == Character_Get( NULL ) )
141: {
142:     state = TEST_STATE_QUIT;
143: }
144: }
145:
146:
147: for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
148: {
149:     IO_Write_U8( board_id, __LINE__, IDI_INTR_BIT_GROUP0 + group, 0x00 );
150: }
151:
152: if ( TEST_STATE_QUIT == state )
153: {
154:     printf( " FAIL" ); error_code = EC_TEST_FAIL;
155: }
156: else
157: {
158:     printf( " PASS" ); error_code = SUCCESS;
159: }
160:
161: printf( "\n" );
162:
163: return error_code;
164: }
```

3.2.1.69 CMD_IDI48_DIN_Test_Interrupt Function

C++

```
static int CMD_IDI48_DIN_Test_Interrupt(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Assumption: IRQs slower than main (see page 239) loop below.

Body Source

```

1: static int CMD_IDI48_DIN_Test_Interrupt( int board_id, int argc, char * argv[] )
2: {
3:     struct idi_dataset * idi_dataset;
4:
5:     int     error_code;
6:     TEST_STATE_ENUM state;
7:     size_t   group;
8:     size_t   bit;
9:     size_t   index;
10:    size_t  irq_count_local;
11:    uint8_t  result;
12:    uint8_t  irq_pending_local;
13:    uint8_t  dig[IDI_DIN_GROUP_QTY];
14:    uint8_t  pending[IDI_DIN_GROUP_QTY];
15:    uint8_t  mask;
16:    BOOL    io_report_backup, io_simulate_backup;
17:    BOOL    din_cmplt[IDI_DIN_QTY];
18: #if defined( DIN_TEST_DEBUG_PRINT )
19:     char        message[64];
20:     int        cp;
21:     int        state_detect;
22: #endif
23: #if defined( __MSDOS__ )
24:     unsigned int    interrupt_number;
25:     switch( board_id )
26:     {
27:         case ID_IDI48:
28:             idi_dataset = ( struct idi_dataset * ) board_definition[board_id].dataset;
29:             break;
30:         default:
31:             return -EC_BOARD;
32:     }
33:
34:     if ( argc > 0 )
35:     {
36:         interrupt_number = (unsigned int) strtol( argv[0], NULL, 0 );
37:     }
38:     else
39:     {
40:         interrupt_number = idi_dataset->irq_number;
41:     }
42: #else
43:
44:     (void)    argc;
45:     (void)    argv;
46:
47:     switch( board_id )
48:     {
49:         case ID_IDI48:
50:             idi_dataset = ( struct idi_dataset * ) board_definition[board_id].dataset;
51:             break;
52:         default:

```

```

53:     return -EC_BOARD;
54: }
55: #endif
56:
57: IO_Write_U8( board_id, __LINE__, IDI_SPI_CONFIG, 0 );
58: io_report_backup = idi_dataset->io_report;
59: io_simulate_backup = idi_dataset->io_simulate;
60:
61: idi_dataset->irq_count_previous = 0;
62: idi_dataset->irq_count = 0;
63:
64: state = TEST_STATE_INIT;
65: irq_pending_local = 0;
66: irq_count_local = 0;
67:
68:
69: for( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
70: {
71:     IDI_DIN_Group_Get( board_id, group, &(dig[group]) );
72:     IO_Write_U8( board_id, __LINE__, IDI_EDGE_GROUP0 + group, 0xFF );
73:     IO_Write_U8( board_id, __LINE__, IDI_CLEAR_GROUP0 + group, 0xFF );
74:     IO_Read_U8( board_id, __LINE__, IDI_PEND_GROUP0 + group, &(pending[group]) );
75: }
76: for ( bit = 0; bit < IDI_DIN_QTY; bit++ ) din_cmplt[bit] = false;
77:
78: printf( "DIN_INTERRUPT:" );
79: for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
80: {
81:     IO_Read_U8( board_id, __LINE__, IDI_PEND_GROUP0 + group, &(pending[group]) );
82:     IDI_DIN_Group_Get( board_id, group, &(dig[group]) );
83: }
84:
85: #if defined( __MSDOS__ )
86:
87: error_code = IOKern_DOS_IRQ_Request( interrupt_number,
88:                                     CMD_IDI48_DIN_Test_Interrupt_Handler,
89:                                     (void *) &board_id
90:                                     );
91: if ( SUCCESS != error_code )
92: {
93:     state = TEST_STATE_QUIT;
94:     goto CMD_IDI48_DIN_Test_Interrupt_Terminate;
95: }
96: #else
97: state = TEST_STATE_QUIT;
98: error_code = -EC_INTERRUPT_UNAVAILABLE;
99: goto CMD_IDI48_DIN_Test_Interrupt_Terminate;
100: #endif
101:
102:
103: for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
104: {
105:     IO_Write_U8( board_id, __LINE__, IDI_INTR_BIT_GROUP0 + group, 0xFF );
106: }
107:
108: idi_dataset->irq_handler_active = true;
109:
110: while ( ( TEST_STATE_QUIT != state ) && ( TEST_STATE_COMPLETE != state ) )
111: {
112:
113:     switch( state )
114:     {
115:         case TEST_STATE_INIT:
116:             for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )

```

```

117:      {
118:        IDI_DIN_Group_Get( board_id, group, &(dig[group]) );
119:      }
120:      if ( true == CMD__IDI48_DIN_Helper_All_Values_False( dig, IDI_DIN_GROUP_QTY ) )
121:      {
122:        state = TEST_STATE_READY;
123:        printf( " READY" );
124:      }
125:      for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ ) IO_Write_U8( board_id,
126: __LINE__, IDI_CLEAR_GROUP0 + group, 0xFF );
127:      break;
128:    case TEST_STATE_READY:
129: #if defined( DIN_TEST_DEBUG_PRINT )
130:      cp = 0;
131:      strcpy( message, "" );
132:      state_detect = 2;
133:      printf( "\nPENDING:" );
134:      for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
135:      {
136:        printf( " 0x%02X", pending[group] );
137:      }
138:    else
139: #endif
140:      state = TEST_STATE_ACTIVE;
141:      break;
142:    case TEST_STATE_ACTIVE:
143:
144:      if ( true == Character_Get( NULL ) ) state = TEST_STATE_QUIT;
145:
146: #if defined( __MSDOS__ )
147:      disable();
148: #endif
149:      irq_count_local = idi_dataset->irq_count;
150:      for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
151:      {
152:        pending[group] = idi_dataset->isr_pending_list[group];
153:        if ( 0 != pending[group] ) irq_pending_local++;
154:      }
155: #if defined( __MSDOS__ )
156:      enable();
157: #endif
158:
159:      if ( irq_count_local != idi_dataset->irq_count_previous )
160:      {
161:        if ( irq_count_local > idi_dataset->irq_count_previous )
162:        {
163:          index = irq_count_local - idi_dataset->irq_count_previous;
164:        }
165:        else
166:        {
167:          index = UINT_MAX - idi_dataset->irq_count_previous;
168:          index = index + irq_count_local + 1;
169:        }
170:        idi_dataset->irq_count_previous = irq_count_local;
171:        irq_count_local = index;
172:      }
173:      else
174:      {
175:        irq_count_local = 0;
176:        irq_pending_local = 0;
177:      }
178:
179:      if ( irq_count_local > 0 )

```

```
180:     {
181: #if defined( DIN_TEST_DEBUG_PRINT )
182: cp += sprintf( &(message[cp]), " {%-2d}", (int) irq_count_local );
183: #endif
184:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
185:     {
186:         if ( 0x00 != pending[group] )
187:         {
188:             mask = 0x01;
189:             for ( bit = 0; bit < IDI_DIN_GROUP_SIZE; bit++ )
190:             {
191:                 index = bit + group * IDI_DIN_GROUP_SIZE;
192:                 if ( 0x00 != (mask & pending[group]) )
193:                 {
194:                     if ( false == din_cmplt[index] )
195:                     {
196:                         din_cmplt[index] = true;
197: #if defined( DIN_TEST_DEBUG_PRINT )
198: cp += sprintf( &(message[cp]), " [%2d]", (int) index );
199: state_detect = 1;
200: #else
201:             printf( "." );
202: #endif
203:                     }
204:                 }
205:                 mask = mask << 1;
206:             }
207:         }
208:     }
209: }
210: #if defined( DIN_TEST_DEBUG_PRINT )
211: if ( state_detect < 3 )
212: {
213:     printf( "\nPENDING:" );
214:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
215:     {
216:         printf( " 0x%02X", pending[group] );
217:     }
218:     printf( "    " );
219:     printf( "%s", message );
220:     cp = 0;
221:     state_detect++;
222: }
223: strcpy( message, "" );
224: #endif
225:     result = true;
226:     for ( bit = 0; bit < IDI_DIN_QTY; bit++ ) result = result & din_cmplt[bit];
227:     if ( true == result ) state = TEST_STATE_COMPLETE;
228:     break;
229: case TEST_STATE_COMPLETE:
230: #if defined( DIN_TEST_DEBUG_PRINT )
231: strcpy( message, "" );
232: printf( "\nPENDING:" );
233:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
234:     {
235:         printf( " 0x%02X", pending[group] );
236:     }
237:     printf( "    " );
238:     printf( "\n" );
239: #endif
240:     break;
241: case TEST_STATE_QUIT:
242:     break;
243: default:
```

```

244:     break;
245: }
246: if ( true == Character_Get( NULL ) )
247: {
248:     state = TEST_STATE_QUIT;
249: }
250: }
251:
252:
253: for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
254: {
255:     IO_Write_U8( board_id, __LINE__, IDI_INTR_BIT_GROUP0 + group, 0x00 );
256: }
257:
258: #if defined( __MSDOS__ )
259: IOKern_DOS_IRQ_Free( idi_dataset->irq_number );
260: idi_dataset->irq_handler_active = false;
261: #endif
262:
263: CMD_IDI48_DIN_Test_Interrupt_Terminate:
264:
265: idi_dataset->io_report = io_report_backup;
266: idi_dataset->io_simulate = io_simulate_backup;
267:
268: if ( TEST_STATE_QUIT == state )
269: {
270:     printf( " FAIL" ); error_code = EC_TEST_FAIL;
271: }
272: else
273: {
274:     printf( " PASS" ); error_code = SUCCESS;
275: }
276:
277: printf( "\n" );
278:
279: return error_code;
280: }
```

3.2.1.70 CMD_IDI48_DIN_Test_Interrupt_Handler Function

C++

```
static irqreturn_t CMD_IDI48_DIN_Test_Interrupt_Handler(int irq, void * dev_id, struct pt_regs * regs);
```

File

idi.c (see page 373)

Description

brief Interrupt Service Routine (ISR). This is a legacy implementation.

Todo

upgrade hardware such that the loop in the ISR can be moved to normal main (see page 239)-line code. Simply reading the IDI_INTR_BY_GROUP ought to be the only hardware interrupt acknowledgement required.

Body Source

```

1: static irqreturn_t CMD__IDI48_DIN_Test_Interrupt_Handler( int irq, void * dev_id, struct
pt_regs * regs )
2: {
3:     size_t group;
4:     uint8_t interrupt_status;
5:     uint8_t mask_group;
6:     uint8_t pending;
7:     uint8_t bank_backup;
8:     uint8_t bank_previous_backup;
9:     int board_id;
10:    struct idi_dataset * idi_dataset;
11:    (void) irq;
12:    (void) dev_id;
13:    (void) regs;
14:
15:    board_id = *((int *) dev_id);
16:    switch( board_id )
17:    {
18:        case ID_IDI48:
19:            idi_dataset = ( struct idi_dataset * ) board_definition[board_id].dataset;
20:            break;
21:        default:
22:            goto CMD__IDI48_DIN_Test_Interrupt_Handler_Completed;
23:    }
24:
25:
26:    bank_previous_backup = idi_dataset->bank_previous;
27:    IO_Read_U8( board_id, __LINE__, IDI_BANK, &bank_backup );
28:
29:    IO_Read_U8( board_id, __LINE__, IDI_INTR_BY_GROUP, &interrupt_status );
30:    mask_group = 0x01;
31:    for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
32:    {
33:        if ( 0 != ( mask_group & interrupt_status ) )
34:        {
35:            IO_Read_U8( board_id, __LINE__, IDI_PEND_GROUP0 + group, &pending );
36:            idi_dataset->isr_pending_list[group] = pending;
37:            IO_Write_U8( board_id, __LINE__, IDI_CLEAR_GROUP0 + group, pending );
38:        }
39:        else
40:        {
41:            idi_dataset->isr_pending_list[group] = 0;
42:        }
43:        mask_group = mask_group << 1;
44:    }
45:
46:    IO_Write_U8( board_id, __LINE__, IDI_BANK, bank_backup );
47:    idi_dataset->bank_previous = bank_previous_backup;
48:    CMD__IDI48_DIN_Test_Interrupt_Handler_Completed:
49:    idi_dataset->irq_count++;
50:    return IRQ_HANDLED;
51: }
```

3.2.1.71 CMD__IDI48_DIN_Test_RE Function

C++

```
static int CMD__IDI48_DIN_Test_RE(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD_IDI48_DIN_Test_RE( int board_id, int argc, char * argv[] )
2: {
3:     int     error_code;
4:     TEST_STATE_ENUM state;
5:     size_t   group;
6:     size_t   bit;
7:     size_t   index;
8:     uint8_t   result;
9:     uint8_t   dig[IDI_DIN_GROUP_QTY];
10:    uint8_t   pending[IDI_DIN_GROUP_QTY];
11:    uint8_t   mask;
12:    BOOL     din_cmplt[IDI_DIN_QTY];
13: #if defined( DIN_TEST_DEBUG_PRINT )
14:     char           message[64];
15:     int      cp;
16:     int      state_detect;
17: #endif
18:     (void)   argc;
19:     (void)   argv;
20:
21:     state = TEST_STATE_INIT;
22:
23:     for( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
24:     {
25:         IDI_DIN_Group_Get( board_id, group, &(dig[group]) );
26:         IO_Write_U8( board_id, __LINE__, IDI_EDGE_GROUP0 + group, 0xFF );
27:         IO_Write_U8( board_id, __LINE__, IDI_CLEAR_GROUP0 + group, 0xFF );
28:         IO_Read_U8( board_id, __LINE__, IDI_PEND_GROUP0 + group, &(pending[group]) );
29:     }
30:     for ( bit = 0; bit < IDI_DIN_QTY; bit++ ) din_cmplt[bit] = false;
31:
32:     printf( "DIN_TEST_RISING_EDGE:" );
33:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
34:     {
35:         IO_Read_U8( board_id, __LINE__, IDI_PEND_GROUP0 + group, &(pending[group]) );
36:         IDI_DIN_Group_Get( board_id, group, &(dig[group]) );
37:         IO_Write_U8( board_id, __LINE__, IDI_INTR_BIT_GROUP0 + group, 0xFF );
38:     }
39:     while ( ( TEST_STATE_QUIT != state ) && ( TEST_STATE_COMPLETE != state ) )
40:     {
41:         switch( state )
42:         {
43:             case TEST_STATE_INIT:
44:                 if ( true == CMD_IDI48_DIN_Helper_All_Values_False( dig, IDI_DIN_GROUP_QTY ) )
45:                 {
46:                     state = TEST_STATE_READY;
47:                     printf( " READY" );

```

```

48:     }
49:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
50:     {
51:         IDI_DIN_Group_Get( board_id, group, &(dig[group]) );
52:         IO_Write_U8( board_id, __LINE__, IDI_CLEAR_GROUP0 + group, 0xFF );
53:     }
54:     break;
55: case TEST_STATE_READY:
56: #if defined( DIN_TEST_DEBUG_PRINT )
57:     cp = 0;
58:     strcpy( message, "" );
59:     state_detect = 2;
60:     printf( "\nPENDING:" );
61:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
62:     {
63:         printf( " 0x%02X", pending[group] );
64:     }
65: #else
66:     printf( " " );
67: #endif
68:     state = TEST_STATE_ACTIVE;
69:     break;
70: case TEST_STATE_ACTIVE:
71:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
72:     {
73:         if ( 0x00 != pending[group] )
74:         {
75:             mask = 0x01;
76:             for ( bit = 0; bit < IDI_DIN_GROUP_SIZE; bit++ )
77:             {
78:                 index = bit + group * IDI_DIN_GROUP_SIZE;
79:                 if ( ( 0x00 != (mask & pending[group]) ) && ( 0x00 != (dig[group] & mask) ) )
80:                 {
81:                     if ( false == din_cmplt[index] )
82:                     {
83:                         din_cmplt[index] = true;
84:                     }
85: #if defined( DIN_TEST_DEBUG_PRINT )
86: cp += sprintf( &(message[cp]), " [%2d]", (int) index );
87: state_detect = 1;
88: #else
89:         printf( "." );
90: #endif
91:         }
92:     }
93:     mask = mask << 1;
94:   }
95:   IO_Write_U8( board_id, __LINE__, IDI_CLEAR_GROUP0 + group, 0xFF );
96: }
97: }
98: #if defined( DIN_TEST_DEBUG_PRINT )
99: if ( state_detect < 3 )
100: {
101:     printf( "\nPENDING:" );
102:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
103:     {
104:         printf( " 0x%02X", pending[group] );
105:     }
106:     printf( "    " );
107:     printf( "%s", message );
108:     cp = 0;
109:     state_detect++;
110: }
111: strcpy( message, "" );

```

```

112: #endif
113:     result = true;
114:     for ( bit = 0; bit < IDI_DIN_QTY; bit++ ) result = result & din_cmplt[bit];
115:     if ( true == result ) state = TEST_STATE_COMPLETE;
116:
117:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
118:     {
119:         IO_Read_U8( board_id, __LINE__, IDI_PEND_GROUP0 + group, &(pending[group]) );
120:         IDI_DIN_Group_Get( board_id, group, &(dig[group]) );
121:     }
122:     break;
123: case TEST_STATE_COMPLETE:
124: #if defined( DIN_TEST_DEBUG_PRINT )
125:     strcpy( message, "" );
126:     printf( "\nPENDING:" );
127:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
128:     {
129:         printf( " 0x%02X", pending[group] );
130:     }
131:     printf( "    " );
132:     printf( "\n" );
133: #endif
134:     break;
135: case TEST_STATE_QUIT:
136:     break;
137: default:
138:     break;
139: }
140: if ( true == Character_Get( NULL ) )
141: {
142:     state = TEST_STATE_QUIT;
143: }
144: }
145:
146:
147: for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
148: {
149:     IO_Write_U8( board_id, __LINE__, IDI_INTR_BIT_GROUP0 + group, 0x00 );
150: }
151:
152: if ( TEST_STATE_QUIT == state )
153: {
154:     printf( " FAIL" ); error_code = EC_TEST_FAIL;
155: }
156: else
157: {
158:     printf( " PASS" ); error_code = SUCCESS;
159: }
160:
161: printf( "\n" );
162:
163: return error_code;
164: }

```

3.2.1.72 CMD_IDI48_DIN_Test_Value Function

C++

```
static int CMD_IDI48_DIN_Test_Value(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```
1: static int CMD__IDI48_DIN_Test_Value( int board_id, int argc, char * argv[] )
2: {
3:     int     error_code;
4:     TEST_STATE_ENUM state;
5:     size_t   group;
6:     size_t   bit;
7:     size_t   index;
8:     uint8_t   result;
9:     uint8_t   dig[IDI_DIN_GROUP_QTY];
10:    uint8_t   dig_prev[IDI_DIN_GROUP_QTY];
11:    uint8_t   group_change;
12:    uint8_t   mask;
13:    BOOL     din_cmplt[IDI_DIN_QTY];
14:    FILE *   out;
15:    struct idi_dataset * dataset;
16:    char *   file_name;
17:    char *   msg;
18:    char   out_default[] = { "stdout" };
19:    (void) argc;
20:    (void) argv;
21:
22:    state = TEST_STATE_INIT;
23:
24:    dataset = NULL;
25:    switch( board_id )
26:    {
27:        case ID_IDI48:
28:            dataset = ( struct idi_dataset * ) board_definition[board_id].dataset;
29:            break;
30:        default:
31:            return -EC_BOARD_TYPE;
32:    }
33:
34:
35:    msg     = &(dataset->message[0]);
36:
37:    out      = stdout;
38:    file_name = out_default;
39:    if ( argc > 0 )
40:    {
41:        size_t wi;
42:        char open_method[4];
43:        wi = 0;
44:        strcpy( open_method, "wt" );
45:        if ( ( 0 == strcasecmp( "--a", argv[wi] ) ) || 
46:             ( 0 == strcasecmp( "--append", argv[wi] ) ) )
47:    {
```

```

48:     wi++;
49:     strcpy( open_method, "at" );
50: }
51:
52: if ( argc > wi )
53: {
54:     out = fopen( argv[wi], open_method );
55:     file_name = argv[wi];
56: }
57: }
58:
59: for( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
60: {
61:     IDI_DIN_Group_Get( board_id, group, &(dig[group]) );
62:     dig_prev[group] = dig[group];
63: }
64: for ( bit = 0; bit < IDI_DIN_QTY; bit++ ) din_cmplt[bit] = false;
65:
66: sprintf( msg, "DIN_TEST_VALUE:" ); Print_Multiple( msg, out );
67: while ( ( TEST_STATE_QUIT != state ) && ( TEST_STATE_COMPLETE != state ) )
68: {
69:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ ) IDI_DIN_Group_Get( board_id,
group, &(dig[group]) );
70:     switch( state )
71:     {
72:         case TEST_STATE_INIT:
73:             if ( true == CMD_IDI48_DIN_Helper_All_Values_False( dig, IDI_DIN_GROUP_QTY ) )
74:             {
75:                 state = TEST_STATE_READY;
76:                 sprintf( msg, "READY:" ); Print_Multiple( msg, out );
77:             }
78:             break;
79:         case TEST_STATE_READY:
80:             sprintf( msg, " " ); Print_Multiple( msg, out );
81:             state = TEST_STATE_ACTIVE;
82:             break;
83:         case TEST_STATE_ACTIVE:
84:             for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
85:             {
86:                 group_change = dig[group] ^ dig_prev[group];
87:                 if ( 0x00 != group_change )
88:                 {
89:                     mask = 0x01;
90:                     for ( bit = 0; bit < IDI_DIN_GROUP_SIZE; bit++ )
91:                     {
92:                         index = bit + group * IDI_DIN_GROUP_SIZE;
93:                         if ( ( 0x00 != ( mask & group_change ) ) && ( false == din_cmplt[index] ) )
94:                         {
95:                             din_cmplt[index] = true;
96:                             sprintf( msg, "." ); Print_Multiple( msg, out );
97:                         }
98:                         mask = mask << 1;
99:                     }
100:                }
101:            }
102:            result = true;
103:            for ( bit = 0; bit < IDI_DIN_QTY; bit++ ) result = result & din_cmplt[bit];
104:            if ( true == result ) state = TEST_STATE_COMPLETE;
105:            break;
106:        case TEST_STATE_COMPLETE:
107:            break;
108:        case TEST_STATE_QUIT:
109:            break;
110:        default:

```

```

111:     break;
112:   }
113:   if ( true == Character_Get( NULL ) )
114:   {
115:     state = TEST_STATE_QUIT;
116:   }
117:   for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ ) dig_prev[group] = dig[group];
118: }
119:
120: {
121:   char * msg = &(dataset->message[0]);
122:   Time_Current_String( msg, MESSAGE_SIZE ); Print_Multiple( msg, out );
123:   sprintf( msg, ":" ); Print_Multiple( msg, out );
124: }
125:
126: if ( TEST_STATE_QUIT == state )
127: {
128:   sprintf( msg, "FAIL" ); error_code = EC_TEST_FAIL;
129: }
130: else
131: {
132:   sprintf( msg, "PASS" ); error_code = SUCCESS;
133: }
134: Print_Multiple( msg, out );
135: sprintf( msg, "\n" );
136: Print_Multiple( msg, out );
137: if ( out != stdout ) fclose( out );
138: return error_code;
139: }
```

3.2.1.73 CMD__IDO48_DO_All Function

C++

```
static int CMD__IDO48_DO_All(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Description

This is function CMD__IDO48_DO_All.

Body Source

```

1: static int CMD__IDO48_DO_All( int board_id, int argc, char * argv[] )
2: {
3:   int error_code;
4:   size_t group;
5:   uint8_t do_grp[6];
6:
7:
8:
9:   for ( group = 0; group < IDO_DO_GROUP_QTY; group++ )
10:  {
11:    error_code = IDO_DO_Group_Get( board_id, group, &(do_grp[group]) );
12:  }
13:
14:  Print_Byt_List( "DO:" ,
15:                 argc,
```

```

16:     argv,
17:     IDO_DO_GROUP_QTY,
18:     do_grp,
19:     stdout
20: );
21: return SUCCESS;
22: }
```

3.2.1.74 CMD__IDO48_DO_Channel Function

C++

```
static int CMD__IDO48_DO_Channel(int board_id, int argc, char * argv[ ]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__IDO48_DO_Channel( int board_id, int argc, char * argv[ ] )
2: {
3:     int error_code;
4:     int channel;
5:     char message[8];
6:     BOOL value;
7:
8:     if ( argc < 1 ) return -EC_NOT_FOUND;
9:
10:    channel = (int) strtol( argv[0], NULL, 0 );
11:
12:    if ( argc < 2 )
13:    {
14:        error_code = IDO_DO_Channel_Get( board_id, channel, &value );
15:        message[0] = value ? '1' : '0';
16:        message[1] = '\0';
17:        printf( "DO%02d: %s\n", channel, message );
18:    }
19:    else
20:    {
21:        value = String_To_Bool( argv[1] );
22:        error_code = IDO_DO_Channel_Set( board_id, channel, value );
23:    }
24:    return SUCCESS;
25: }
```

3.2.1.75 CMD__IDO48_DO_Group Function

C++

```
static int CMD__IDO48_DO_Group(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```
1: static int CMD__IDO48_DO_Group( int board_id, int argc, char * argv[] )
2: {
3:     int    error_code;
4:     int    group;
5:     int    group_start;
6:     int    group_end;
7:
8:     BOOL   read_cycle;
9:     uint8_t do_grp[IDO_DO_GROUP_QTY];
10:
11:
12:    if ( argc < 1 )
13:    {
14:        group_start = 0;
15:        group_end   = IDO_DO_GROUP_QTY - 1;
16:
17:        read_cycle = true;
18:    }
19:    else if ( argc > 1 )
20:    {
21:        group_start = (int) strtol( argv[0], NULL, 0 );
22:        if ( group_start >= IDO_DO_GROUP_QTY ) return -EC_CHANNEL;
23:        group_end   = IDO_DO_GROUP_QTY - 1;
24:
25:        read_cycle = false;
26:    }
27:    else
28:    {
29:        group_start = (int) strtol( argv[0], NULL, 0 );
30:        if ( group_start >= IDO_DO_GROUP_QTY ) return -EC_CHANNEL;
31:        group_end   = group_start + 1;
32:
33:        read_cycle = true;
34:    }
35:
36:    if ( true == read_cycle )
37:    {
38:        for ( group = group_start; group < group_end; group++ )
39:    {
```

```

40:     error_code = IDO_DO_Group_Get( board_id, group, &(do_grp[group]) );
41: }
42:
43: printf( "DO_GROUP:" );
44: for ( group = group_start; group < group_end; group++ )
45: {
46:     printf( " 0x%02X", ((int) do_grp[group]) );
47: }
48: printf( "\n" );
49: }
50: else
51: {
52:     int index;
53:     group_end = group_start;
54:     group      = group_start;
55:     for ( index = 1; index < argc; index++ )
56:     {
57:         do_grp[group] = (int) strtol( argv[index], NULL, 0 );
58:         group++;
59:         group_end++;
60:         if ( group >= IDO_DO_GROUP_QTY ) break;
61:     }
62:
63:     for ( group = group_start; group < group_end; group++ )
64:     {
65:         error_code = IDO_DO_Group_Set( board_id, group, do_grp[group] );
66:     }
67: }
68: return SUCCESS;
69: }
```

3.2.1.76 CMD__IDO48_DO_ID Function

C++

```
static int CMD__IDO48_DO_ID(int board_id, int argc, char * argv[ ]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

brief

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__IDO48_DO_ID( int board_id, int argc, char * argv[ ] )
2: {
3:     uint16_t id;
4:     (void) argc;
5:     (void) argv;
```

```
6: IDO_DOUT_ID_Get( board_id, &id );
7: printf( "IDO48 DOUT ID: 0x%04X\n", id );
8: return SUCCESS;
9: }
```

3.2.1.77 CMD__IDO48_DOUT_Test_Alternate Function

C++

```
static int CMD__IDO48_DOUT_Test_Alternate(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```
1: static int CMD__IDO48_DOUT_Test_Alternate( int board_id, int argc, char * argv[ ] )
2: {
3:     TEST_STATE_ENUM state;
4:     size_t group;
5: #if defined( __MSDOS__ )
6:     size_t delay_normal_ms;
7:     size_t delay_inverted_ms;
8: #endif
9:     uint8_t pattern;
10:    uint8_t dog[IDO_DO_GROUP_QTY];
11:    char * mode_string[] = { "binary" };
12:
13:    state = TEST_STATE_INIT;
14: #if defined( __MSDOS__ )
15:     delay_normal_ms = 100;
16:     delay_inverted_ms = 20;
17: #endif
18:    pattern = 0x55;
19:
20:    if ( argc > 0 )
21:    {
22:        pattern = (uint8_t) strtol( argv[0], NULL, 0 );
23:    }
24:
25:    if ( argc > 1 )
26:    {
27:        if ( 0 == strcmpi( "time", argv[1] ) )
28:        {
29:            if ( argc > 2 )
30:            {
31: #if defined( __MSDOS__ )
32:                 delay_normal_ms = (size_t) strtol( argv[2], NULL, 0 );
33: #endif
34:            }
35:        else if ( argc > 3 )
```

```

36: {
37: #if defined( __MSDOS__ )
38:     delay_inverted_ms = (size_t) strtol( argv[3], NULL, 0 );
39: #endif
40: }
41: }
42: }
43:
44:
45: for( group = 0; group < IDO_DO_GROUP_QTY; group++ )
46: {
47:     IDO_DO_Group_Set( board_id, group, 0x00 );
48:     dog[group] = 0x00;
49: }
50:
51: state = TEST_STATE_ACTIVE;
52: printf( "IDO48 Alternating sequence testing (press any key to quit):" );
53: while ( TEST_STATE_ACTIVE == state )
54: {
55:     for( group = 0; group < IDO_DO_GROUP_QTY; group++ )
56:     {
57:         IDO_DO_Group_Set( board_id, group, pattern );
58:         dog[group] = pattern;
59:     }
60:
61:     for( group = 0; group < IDO_DO_GROUP_QTY; group++ )
62:     {
63:         IDO_DO_Group_Get( board_id, group, &(dog[group]) );
64:     }
65:     Print_Byte_List( "DO:",
66:                     1,
67:                     mode_string,
68:                     IDO_DO_GROUP_QTY,
69:                     dog,
70:                     stdout
71:                     );
72: #if defined( __MSDOS__ )
73:     delay( delay_normal_ms );
74: #endif
75:
76:     for( group = 0; group < IDO_DO_GROUP_QTY; group++ )
77:     {
78:         IDO_DO_Group_Set( board_id, group, ~pattern );
79:         dog[group] = ~pattern;
80:     }
81:
82:     for( group = 0; group < IDO_DO_GROUP_QTY; group++ )
83:     {
84:         IDO_DO_Group_Get( board_id, group, &(dog[group]) );
85:     }
86:     Print_Byte_List( "DO:",
87:                     1,
88:                     mode_string,
89:                     IDO_DO_GROUP_QTY,
90:                     dog,
91:                     stdout
92:                     );
93: #if defined( __MSDOS__ )
94:     delay( delay_inverted_ms );
95: #endif
96:     if ( true == Character_Get( NULL ) ) state = TEST_STATE_QUIT_USER;
97: }
98:
99:
```

```

100: for( group = 0; group < IDO_DO_GROUP_QTY; group++ )
101: {
102:     IDO_DO_Group_Set( board_id, group, 0x00 );
103: }
104: return SUCCESS;
105: }
```

3.2.1.78 CMD__IDO48_DOUT_Test_One_Hot Function

C++

```
static int CMD__IDO48_DOUT_Test_One_Hot(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__IDO48_DOUT_Test_One_Hot( int board_id, int argc, char * argv[ ] )
2: {
3:     TEST_STATE_ENUM state;
4:     size_t group;
5:     size_t channel;
6: #if defined( __MSDOS__ )
7:     size_t delay_on_ms;
8:     size_t delay_off_ms;
9: #endif
10:    uint8_t dog[IDO_DO_GROUP_QTY];
11:    char * mode_string[] = { "binary" };
12:
13:    state = TEST_STATE_INIT;
14: #if defined( __MSDOS__ )
15:     delay_on_ms = 100;
16:     delay_off_ms = 20;
17: #endif
18:    if ( argc > 0 )
19:    {
20: #if defined( __MSDOS__ )
21:     delay_on_ms = (size_t) strtol( argv[0], NULL, 0 );
22: #endif
23:    }
24:    else if ( argc > 1 )
25:    {
26: #if defined( __MSDOS__ )
27:     delay_off_ms = (size_t) strtol( argv[1], NULL, 0 );
28: #endif
29:    }
30:
31:
32:    for( group = 0; group < IDO_DO_GROUP_QTY; group++ )
33:    {
```

```

34:     dog[group] = 0x00;
35:     IDO_DO_Group_Set( board_id, group, dog[group] );
36: }
37:
38: state = TEST_STATE_ACTIVE;
39: printf( "IDO48 One-hot sequence testing (press any key to quit):" );
40: channel = 0;
41: while ( TEST_STATE_ACTIVE == state )
42: {
43:     IDO_DO_Channel_Set( board_id, channel, true );
44:
45:     for( group = 0; group < IDO_DO_GROUP_QTY; group++ )
46:     {
47:         IDO_DO_Group_Get( board_id, group, &(dog[group]) );
48:     }
49:     Print_Byte_List( "DO:",
50:                     1,
51:                     mode_string,
52:                     IDO_DO_GROUP_QTY,
53:                     dog,
54:                     stdout
55:                     );
56: #if defined( __MSDOS__ )
57:     delay( delay_on_ms );
58: #endif
59:
60:     IDO_DO_Channel_Set( board_id, channel, false );
61: #if defined( __MSDOS__ )
62:     delay( delay_off_ms );
63: #endif
64:
65:     for( group = 0; group < IDO_DO_GROUP_QTY; group++ )
66:     {
67:         IDO_DO_Group_Get( board_id, group, &(dog[group]) );
68:     }
69:     Print_Byte_List( "DO:",
70:                     1,
71:                     mode_string,
72:                     IDO_DO_GROUP_QTY,
73:                     dog,
74:                     stdout
75:                     );
76:
77:     if ( true == Character_Get( NULL ) ) state = TEST_STATE_QUIT_USER;
78:     channel++;
79:     if ( channel >= IDO_DO_QTY ) channel = 0;
80: }
81:
82:
83: for( group = 0; group < IDO_DO_GROUP_QTY; group++ )
84: {
85:     IDO_DO_Group_Set( board_id, group, 0x00 );
86: }
87: return SUCCESS;
88: }
```

3.2.1.79 CMD__IDO48_IDI48_Loopback Function

C++

```
static int CMD__IDO48_IDI48_Loopback(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Description

brief

Body Source

```
1: static int CMD__IDO48_IDI48_Loopback( int board_id, int argc, char * argv[] )
2: {
3:     int     error_code;
4:     struct  ido_dataset * dataset;
5:     FILE   * out;
6:     char   * file_name;
7:     TEST_STATE_ENUM state;
8:     size_t   group;
9:     size_t   channel;
10:    BOOL    di_value;
11:    BOOL    do_value;
12: #if defined( __MSDOS__ )
13:    size_t   wait_ms;
14: #endif
15:    uint8_t   do_list[IDO_DO_GROUP_QTY];
16:    uint8_t   di_list[IDI_DIN_GROUP_QTY];
17:    uint8_t   dog[IDO_DO_GROUP_QTY];
18:    char   * mode_string[] = { "binary" };
19:    char   * out_default[] = { "stdout" };
20:    char   * msg;
21:
22:    state = TEST_STATE_INIT;
23:    dataset = NULL;
24:    switch( board_id )
25:    {
26:        case ID_IDO48:
27:            dataset = ( struct ido_dataset * ) board_definition[board_id].dataset;
28:            break;
29:        default:
30:            return -EC_BOARD_TYPE;
31:    }
32:
33: #if defined( __MSDOS__ )
34:    wait_ms = 100;
35: #endif
36:    out      = stdout;
37:    file_name = out_default;
38:    if ( argc > 0 )
39:    {
40:        size_t   wi;
41:        char   open_method[4];
42:        wi = 0;
43:        strcpy( open_method, "wt" );
44:        if ( ( 0 == strcasecmp( "--a", argv[wi] ) ) || ||
45:             ( 0 == strcasecmp( "--append", argv[wi] ) ) )
46:        {
47:            wi++;
48:            strcpy( open_method, "at" );
49:        }
50:
51:        if ( argc > wi )
52:    {
```

```
53:     out = fopen( argv[wi], open_method );
54:     file_name = argv[wi];
55: }
56: }
57:
58:
59: msg = &(dataset->message[0]);
60: sprintf( msg, "===== " );
61: Print_Multiple( msg, out );
62: Time_Current_String( msg, MESSAGE_SIZE );
63: Print_Multiple( msg, out );
64: sprintf( msg, "\nIDO48-IDI48 loopback test: %s", file_name );
65: Print_Multiple( msg, out );
66:
67:
68:
69: sprintf( msg, "nIDO48-IDI48 loopback test: %s", file_name );
70: Print_Multiple( msg, out );
71:
72:
73: for( group = 0; group < IDO_DO_GROUP_QTY; group++ )
74: {
75:     do_list[group] = 0x00;
76:     IDO_DO_Group_Set( board_id, group, do_list[group] );
77: #if defined( __MSDOS__ )
78:     delay( wait_ms );
79: #endif
80:     IDI_DIN_Group_Get( ID_IDI48, group, &(di_list[group]) );
81: }
82: error_code = IDO48_IDI48_Loopback_Test( ID_IDI48, board_id, di_list, do_list,
IDO_DO_GROUP_QTY );
83: if ( error_code < 0 )
84: {
85:     state = TEST_STATE_QUIT;
86: }
87:
88: state = TEST_STATE_ACTIVE;
89: channel = 0;
90: while ( TEST_STATE_ACTIVE == state )
91: {
92:
93:
94:     do_value = true;
95:     IDO_DO_Channel_Set( board_id, channel, do_value );
96: #if defined( __MSDOS__ )
97:     delay( wait_ms );
98: #endif
99:     IDI_DIN_Channel_Get( ID_IDI48, channel, &di_value );
100:
101: if ( do_value != di_value )
102: {
103:     state = TEST_STATE_QUIT_ERROR;
104: }
105: error_code = IDO48_IDI48_Loopback_Test( ID_IDI48, board_id, di_list, do_list,
IDO_DO_GROUP_QTY );
106: if ( error_code < 0 )
107: {
108:     state = TEST_STATE_QUIT_ERROR;
109: }
110:
111:
112: do_value = false;
113: IDO_DO_Channel_Set( board_id, channel, do_value );
114: #if defined( __MSDOS__ )
115:     delay( wait_ms );
116: #endif
117: IDI_DIN_Channel_Get( ID_IDI48, channel, &di_value );
```

```
118:
119:     if ( do_value != di_value )
120:     {
121:         state = TEST_STATE_QUIT_ERROR;
122:     }
123:     error_code = IDO48_IDI48_Loopback_Test( ID_IDI48, board_id, di_list, do_list,
IDO_DO_GROUP_QTY );
124:     if ( error_code < 0 )
125:     {
126:         state = TEST_STATE_QUIT_ERROR;
127:     }
128:
129:
130:     {
131:         size_t index;
132:         uint8_t do_group[IDO_DO_GROUP_QTY];
133:
134:         for ( index = 0; index < IDO_DO_GROUP_QTY; index++ )
135:         {
136:             do_group[index] = 0xFF;
137:             IDO_DO_Group_Set( board_id, index, do_group[index] );
138:         }
139: #if defined( __MSDOS__ )
140:     delay( wait_ms );
141: #endif
142:     error_code = IDO48_IDI48_Loopback_Test( ID_IDI48, board_id, di_list, do_list,
IDO_DO_GROUP_QTY );
143:     if ( error_code < 0 )
144:     {
145:         sprintf( msg, "All high fail" );
146:         Print_Multiple( msg, out );
147:         state = TEST_STATE_QUIT_ERROR;
148:     }
149: else
150: {
151:     for ( index = 0; index < IDO_DO_GROUP_QTY; index++ )
152:     {
153:         do_group[index] = 0x00;
154:         IDO_DO_Group_Set( board_id, index, do_group[index] );
155:     }
156: }
157: }
158:
159:
160: if ( true == Character_Get( NULL ) ) state = TEST_STATE_QUIT_USER;
161:
162: if ( TEST_STATE_ACTIVE == state )
163: {
164:     channel++;
165:     sprintf( msg, "." );
166:     Print_Multiple( msg, out );
167:     if ( channel >= IDO_DO_QTY ) state = TEST_STATE_COMPLETE;
168: }
169: }
170:
171:
172: sprintf( msg, "[PATTERN_INVERT]" );
173: Print_Multiple( msg, out );
174: channel = 0;
175: if ( TEST_STATE_COMPLETE == state ) state = TEST_STATE_ACTIVE;
176: while ( TEST_STATE_ACTIVE == state )
177: {
178:     {
179:         size_t index;
```

```

180:
181:     for ( index = 0; index < IDO_DO_GROUP_QTY; index++ )
182:     {
183:         IDO_DO_Group_Set( board_id, index, 0xFF );
184:     }
185: #if defined( __MSDOS__ )
186:     delay( wait_ms );
187: #endif
188:     error_code = IDO48_IDI48_Loopback_Test( ID_IDI48, board_id, di_list, do_list,
189: IDO_DO_GROUP_QTY );
190:     if ( error_code < 0 )
191:     {
192:         sprintf( msg, "All high fail" );
193:         Print_Multiple( msg, out );
194:         state = TEST_STATE_QUIT_ERROR;
195:     }
196:
197:
198:     do_value = false;
199:     IDO_DO_Channel_Set( board_id, channel, do_value );
200: #if defined( __MSDOS__ )
201:     delay( wait_ms );
202: #endif
203:     IDI_DIN_Channel_Get( ID_IDI48, channel, &di_value );
204:
205:     if ( do_value != di_value )
206:     {
207:         state = TEST_STATE_QUIT_ERROR;
208:     }
209:     error_code = IDO48_IDI48_Loopback_Test( ID_IDI48, board_id, di_list, do_list,
210: IDO_DO_GROUP_QTY );
211:     if ( error_code < 0 )
212:     {
213:         state = TEST_STATE_QUIT_ERROR;
214:     }
215:
216:     do_value = true;
217:     IDO_DO_Channel_Set( board_id, channel, do_value );
218: #if defined( __MSDOS__ )
219:     delay( wait_ms );
220: #endif
221:     IDI_DIN_Channel_Get( ID_IDI48, channel, &di_value );
222:
223:     if ( do_value != di_value )
224:     {
225:         state = TEST_STATE_QUIT_ERROR;
226:     }
227:     error_code = IDO48_IDI48_Loopback_Test( ID_IDI48, board_id, di_list, do_list,
228: IDO_DO_GROUP_QTY );
229:     if ( error_code < 0 )
230:     {
231:         state = TEST_STATE_QUIT_ERROR;
232:     }
233:     if ( true == Character_Get( NULL ) ) state = TEST_STATE_QUIT_USER;
234:
235:     if ( TEST_STATE_ACTIVE == state )
236:     {
237:         channel++;
238:         sprintf( msg, "." );
239:         Print_Multiple( msg, out );
240:         if ( channel >= IDO_DO_QTY ) state = TEST_STATE_COMPLETE;

```

```
241:     }
242: }
243:
244:
245: {
246:     size_t index;
247:     for ( index = 0; index < IDO_DO_GROUP_QTY; index++ )
248:     {
249:         IDO_DO_Group_Set( board_id, index, 0x00 );
250:     }
251: #if defined( __MSDOS__ )
252: delay( wait_ms );
253: #endif
254: }
255:
256:
257: sprintf( msg, "\n" );
258: Print_Multiple( msg, out );
259: switch( state )
260: {
261:     case TEST_STATE_COMPLETE:
262:         sprintf( msg, "PASS\n" );
263:         Print_Multiple( msg, out );
264:         error_code = SUCCESS;
265:         break;
266:     case TEST_STATE_QUIT_ERROR:
267:         sprintf( msg, "FAIL at channel %d\n", (int) channel );
268:         Print_Multiple( msg, out );
269:         error_code = EC_TEST_FAIL;
270:
271:     for( group = 0; group < IDO_DO_GROUP_QTY; group++ )
272:     {
273:         IDO_DO_Group_Get( board_id, group, &(dog[group]) );
274:     }
275:     Print_Byte_List( "DO:",
276:                     1,
277:                     mode_string,
278:                     IDO_DO_GROUP_QTY,
279:                     dog,
280:                     stdout
281:                     );
282:
283:     if ( out != stdout )
284:     {
285:         Print_Byte_List( "DO:",
286:                         1,
287:                         mode_string,
288:                         IDO_DO_GROUP_QTY,
289:                         dog,
290:                         out
291:                         );
292:     }
293:     break;
294:     case TEST_STATE_QUIT_USER:
295:         sprintf( msg, "CANCEL BY USER at channel %d\n", (int) channel );
296:         Print_Multiple( msg, out );
297:         error_code = EC_TEST_FAIL;
298:         break;
299:     default:
300:         break;
301:     }
302:     if ( out != stdout ) fclose( out );
303:     return error_code;
304: }
```

3.2.1.80 CMD__IDO48_Test Function

C++

```
static int CMD__IDO48_Test(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Cycles through all channels one at a time in a one-hot situation.

Body Source

```

1: static int CMD__IDO48_Test( int board_id, int argc, char * argv[] )
2: {
3:     int     error_code;
4:     int     index;
5:     int     argc_new;
6:     char ** argv_new;
7:     char *    endptr;
8:
9:     if ( argc < 1 ) return -EC_NOT_FOUND;
10:
11:    error_code = -EC_SYNTAX;
12:
13:    strtol( argv[0], &endptr, 0 );
14:    if ( argv[0] != endptr )
15:    {
16:        error_code = (* cmd_ido48_test[0].cmd_fnc )( board_id, argc, argv );
17:    }
18:    else
19:    {
20:        index = 0;
21:        while ( NULL != cmd_ido48_test[index].cmd_fnc )
22:        {
23:            if ( 0 == strcasecmp( cmd_ido48_test[index].name, argv[0] ) )
24:            {
25:                argv_new = &(argv[1]);
26:                argc_new = argc - 1;
27:                if ( 0 == argc_new ) argv_new = NULL;
28:                error_code = (* cmd_ido48_test[index].cmd_fnc )( board_id, argc_new, argv_new );
29:                break;
30:            }
31:            index++;
32:        }
33:    }
34:    return error_code;
35: }
```

3.2.1.81 CMD__Main_AnalogStick_CS Function

C++

```
static int CMD__Main_AnalogStick_CS(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```
1: static int CMD__Main_AnalogStick_CS( int board_id, int argc, char * argv[] )
2: {
3:     struct board_dataset * dataset;
4:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
5:
6:     if ( ( argc < 1 ) || ( NULL == argv ) )
7:     {
8:         printf( "aics          = " );
9:         if ( true == dataset->as_cs_default ) printf( "1" );
10:        else                                printf( "0" );
11:        printf( "  (Analog Stick SPI Chip Select)\n" );
12:    }
13:    else
14:    {
15:        dataset->as_cs_default = String_To_Bool( argv[0] );
16:        printf( "OK\n" );
17:    }
18:    return SUCCESS;
19: }
```

3.2.1.82 CMD__Main_Base Function

C++

```
static int CMD__Main_Base(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__Main_Base( int board_id, int argc, char * argv[] )
2: {
3:     struct board_dataset * dataset;
4:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
5:
6:     if ( ( argc < 1 ) || ( NULL == argv ) )
7:     {
8:         printf( "base           = 0x%04X\n", dataset->base_address );
9:     }
10:    else
11:    {
12:        dataset->base_address = (uint16_t) strtol( argv[0], NULL, 0 );
13:        printf( "OK\n" );
14:    }
15:    return SUCCESS;
16: }
```

3.2.1.83 CMD__Main_FRAM_CS Function

C++

```
static int CMD__Main_FRAM_CS(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__Main_FRAM_CS( int board_id, int argc, char * argv[] )
2: {
3:     struct board_dataset * dataset;
4:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
5:
6:     if ( ( argc < 1 ) || ( NULL == argv ) )
7:     {
8:         printf( "framcs      = " );
9:         if ( true == dataset->fram_cs_default ) printf( "1" );
10:        else                         printf( "0" );
11:        printf( " (FRAM SPI Chip Select)\n" );
12:    }
13:    else
14:    {
15:        dataset->fram_cs_default = String_To_Bool( argv[0] );
```

```
16:     printf( "OK\n" );
17: }
18: return SUCCESS;
19: }
```

3.2.1.84 CMD__Main_I_Count Function

C++

```
static int CMD__Main_I_Count(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Number of interrupt cycles. If "loop" is used, then the count can be terminated by pressing any key on the keyboard input.

Body Source

```
1: static int CMD__Main_I_Count( int board_id, int argc, char * argv[ ] )
2: {
3:     struct idi_dataset * idi_dataset = NULL;
4:
5:     switch( board_id )
6:     {
7:         case ID_IDI48:
8:             idi_dataset = ( struct idi_dataset * ) board_definition[board_id].dataset;
9:             break;
10:        }
11:
12:        if ( ( argc < 1 ) || ( NULL == argv ) )
13:        {
14:            if ( NULL == idi_dataset ) return SUCCESS;
15: #if defined( __MSDOS__ )
16:             printf( "iqty                 = %d\n", idi_dataset->irq_quantity );
17: #else
18:             printf( "iqty                 = %lu\n", idi_dataset->irq_quantity );
19: #endif
20:        }
21:        else
22:        {
23:            if ( NULL == idi_dataset ) return -EC_BOARD_TYPE;
24:            idi_dataset->irq_quantity = (size_t) strtol( argv[0], NULL, 0 );
25:            printf( "OK\n" );
26:        }
27:        return SUCCESS;
28: }
```

3.2.1.85 CMD__Main_Init_Reg Function

C++

```
static int CMD__Main_Init_Reg(int board_id, int argc, char * argv[ ]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__Main_Init_Reg( int board_id, int argc, char * argv[ ] )
2: {
3:     int index;
4:     int error_code;
5:     struct board_dataset * dataset;
6:     struct reg_definition * definition;
7:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
8:     definition = ( struct reg_definition * ) board_definition[board_id].definition;
9:
10:    if ( argc < 1 )
11:    {
12:        for ( index = 0; index < REGS_INIT_QTY; index++ )
13:        {
14:            printf ( "reg_init[%2d]: ", index );
15:            if ( dataset->reg_init[index].used )
16:            {
17:                if ( IDI_UNDEFINED != dataset->reg_init[index].location )
18:                {
19:                    row = REG_LOCATION_LOGICAL_GET( location );
20:                    printf( "reg = %s, value = 0x%02X", definition[row].acronym, value );
21:                }
22:            else
23:            {
24:                printf( "address = 0x%04X, value = 0x%02X", address, value );
25:            }
26:        }
27:    else
28:    {
29:        printf( "unused" );
30:    }
31:    }
32:    printf( "\n" );
33: }
34: else if ( argc > 1 )
35: {
36:    for ( index = 0; index < REGS_INIT_QTY; index++ )
37:    {
38:        if ( IDI_UNDEFINED != dataset->reg_init[index].location )
39:        {
```

```

40:     if ( SUCCESS == Register_Acronym_To_Row( board_id, argv[0], &index ) )
41:     {
42:
43:     }
44:     else
45:     {
46:
47:     }
48:   }
49: }
50:
51:
52: if ( argc > 0 )
53: {
54:
55:
56: }
57:
58: return SUCCESS;
59: }
```

3.2.1.86 CMD__Main_IO_Behavior Function

C++

```
static int CMD__Main_IO_Behavior(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__Main_IO_Behavior( int board_id, int argc, char * argv[] )
2: {
3:   struct board_dataset * dataset;
4:   dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
5:
6:   if ( ( argc < 1 ) || ( NULL == argv ) )
7:   {
8:     printf( "IO Simulate      = %s\n", dataset->io_simulate ? "true" : "false" );
9:     printf( "IO Report        = %s\n", dataset->io_report    ? "true" : "false" );
10:
11:   else if ( argc > 1 )
12:   {
13:     if       ( 0 == strcmpi( "simulate", argv[0] ) )
14:     {
15:       dataset->io_simulate = String_To_Bool( argv[1] );
16:     }
17:     else if  ( 0 == strcmpi( "report", argv[0] ) )
18:     {
19:       dataset->io_report = String_To_Bool( argv[1] );
```

```

20:    }
21:  else
22:  {
23:
24:  }
25:  printf( "OK\n" );
26: }
27: else
28: {
29:  if      ( 0 == strcasecmp( "simulate", argv[0] ) )
30:  {
31:    printf( "IO Simulate      = %s\n", dataset->io_simulate ? "true" : "false" );
32:  }
33: else if   ( 0 == strcasecmp( "report", argv[0] ) )
34:  {
35:    printf( "IO Report       = %s\n", dataset->io_report    ? "true" : "false" );
36:  }
37: }
38: return SUCCESS;
39: }
```

3.2.1.87 CMD__Main_Irq_Number Function

C++

```
static int CMD__Main_Irq_Number(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__Main_Irq_Number( int board_id, int argc, char * argv[] )
2: {
3:  struct idi_dataset * idi_dataset = NULL;
4:
5:  switch( board_id )
6:  {
7:    case ID_IDI48:
8:      idi_dataset = ( struct idi_dataset * ) board_definition[board_id].dataset;
9:      break;
10: }
11:
12: if ( ( argc < 1 ) || ( NULL == argv ) )
13: {
14:  if ( NULL == idi_dataset ) return SUCCESS;
15: #if defined( __MSDOS__ )
16:  printf( "irq           = %u\n", idi_dataset->irq_number );
17: #else
18:  printf( "irq           = %u\n", idi_dataset->irq_number );
19: #endif
```

```

20: }
21: else
22: {
23:     if ( NULL == idi_dataset ) return -EC_BOARD_TYPE;
24:     idi_dataset->irq_number = (unsigned int) strtol( argv[0], NULL, 0 );
25:     printf( "OK\n" );
26: }
27: return SUCCESS;
28: }
```

3.2.1.88 CMD__Main_Mode_Jumpers Function

C++

```
static int CMD__Main_Mode_Jumpers(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Mode jumper settings. 0 = 00 =

Body Source

```

1: static int CMD__Main_Mode_Jumpers( int board_id, int argc, char * argv[] )
2: {
3:     struct board_dataset * ds = NULL;
4:
5:     switch( board_id )
6:     {
7:         case ID_IDI48:
8:             ds = ( struct board_dataset * ) board_definition[board_id].dataset;
9:             break;
10:        case ID_IDO48:
11:            ds = ( struct board_dataset * ) board_definition[board_id].dataset;
12:            break;
13:    }
14:
15:    if ( ( argc < 1 ) || ( NULL == argv ) )
16:    {
17:        if ( NULL == ds ) return SUCCESS;
18:        printf( "mode                = " );
19:        switch( ds->mode_jumpers )
20:        {
21:            case MODE_JUMPERS_0: printf( "0 (legacy)" ); break;
22:            case MODE_JUMPERS_1: printf( "1                 " ); break;
23:            case MODE_JUMPERS_2: printf( "2                 " ); break;
24:            case MODE_JUMPERS_3: printf( "3                 " ); break;
25:        }
26:        printf( ". Assumed mode jumper settings.\n" );
27:    }
28:    else
29:    {
30:        if ( NULL == ds ) return -EC_BOARD_TYPE;
```

```

31: if ( 0 == strcasecmp( "legacy", argv[0] ) )
32: {
33:     ds->mode_jumpers = MODE_JUMPERS_0;
34: }
35: else
36: {
37:     ds->mode_jumpers = (MODE_JUMPERS_ENUM) strtol( argv[0], NULL, 0 );
38: }
39: printf( "OK\n" );
40: }
41: return SUCCESS;
42: }
```

3.2.1.89 CMD__Main_Trace Function

C++

```
static int CMD__Main_Trace(int board_id, int argc, char* argv[ ]);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int CMD__Main_Trace( int board_id,  int argc,  char* argv[] )
2: {
3:     int    error_code;
4:     int    index;
5:     int    argc_new;
6:     char ** argv_new;
7:     (void) board_id;
8:
9:     error_code = -EC_SYNTAX;
10:
11:    if ( ( argc < 1 ) || ( NULL == argv ) )
12:    {
13:        index = 0;
14:        while ( NULL != cmd_trace[index].cmd_fnc )
15:        {
16:            printf( "trace." );
17:            argv_new = &(argv[1]);
18:            argc_new = argc - 1;
19:            if ( 0 == argc_new ) argv_new = NULL;
20:            error_code = (* cmd_trace[index].cmd_fnc )( board_id, argc_new, argv_new );
21:            index++;
22:        }
23:    }
24:    else
25:    {
26:        index = 0;
27:        while ( NULL != cmd_trace[index].cmd_fnc )
28:        {
29:            if ( 0 == strcasecmp( cmd_trace[index].name, argv[0] ) )
30:            {
31:                argv_new = &(argv[1]);
32:                argc_new = argc - 1;
```

```

33:     if ( 0 == argc_new ) argv_new = NULL;
34:     error_code = (* cmd_trace[index].cmd_fnc )( board_id, argc_new, argv_new );
35:     break;
36:   }
37:   index++;
38: }
39: }
40: return error_code;
41: }
```

3.2.1.90 CMD__SPI_Commit Function

C++

```
static int CMD__SPI_Commit(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__SPI_Commit( int board_id, int argc, char * argv[] )
2: {
3:     int     error_code;
4:     uint8_t commit;
5:
6:     if      ( argc < 1 )
7:     {
8:         uint8_t commit_status;
9:         SPI_Commit_Get( board_id, &commit_status );
10:        printf( "commit status = %s\n", ( 0 == commit_status ) ? "false" : "true" );
11:    }
12:    else if ( String_To_Bool( argv[0] ) ) commit = 0x01;
13:    else                                commit = 0x00;
14:
15:    error_code = SPI_Commit( board_id, commit );
16:    if ( SUCCESS != error_code ) return error_code;
17:
18:    printf( "OK\n" );
19:    return SUCCESS;
20: }
```

3.2.1.91 CMD__SPI_Config_Chip_Select_Behavior Function

C++

```
static int CMD__SPI_Config_Chip_Select_Behavior(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__SPI_Config_Chip_Select_Behavior( int board_id, int argc, char * argv[ ] )
2: {
3:     int     error_code;
4:
5:     struct    spi_cfg cfg;
6:
7:
8:     error_code = SPI_Configuration_Get( board_id, &cfg );
9:     if ( SUCCESS != error_code ) return error_code;
10:
11:    if ( argc < 1 )
12:    {
13:        printf( "SPI CSB: " );
14:        switch ( cfg.chip_select_behavior )
15:        {
16:            case CSB_SOFTWARE: printf( "software" ); break;
17:            case CSB_BUFFER: printf( "buffer" ); break;
18:            case CSB_uint8_t: printf( "uint8_t" ); break;
19:            case CSB_uint16_t: printf( "uint16_t" ); break;
20:            default:           printf( "undefined" ); break;
21:        }
22:        printf( "\n" );
23:    }
24:    else
25:    {
26:        if      ( 0 == strcasecmp( "software", argv[0] ) ) cfg.chip_select_behavior = 0;
27:        else if ( 0 == strcasecmp( "buffer", argv[0] ) ) cfg.chip_select_behavior = 1;
28:        else if ( 0 == strcasecmp( "uint8_t", argv[0] ) ) cfg.chip_select_behavior = 2;
29:        else if ( 0 == strcasecmp( "uint16_t", argv[0] ) ) cfg.chip_select_behavior = 3;
30:        else
31:        {
32:            cfg.chip_select_behavior = (SPI_CSB_ENUM) strtol( argv[0], NULL, 0 );
33:            switch ( cfg.chip_select_behavior )
34:            {
35:                case CSB_SOFTWARE:
36:                case CSB_BUFFER:
37:                case CSB_uint8_t:
38:                case CSB_uint16_t:
39:                    break;
40:                default:
41:                    error_code = -EC_SPI_CSB_OUT_OF_RANGE;
42:                    break;
43:            }
44:        }
45:        if ( SUCCESS != error_code ) return error_code;
46:
47:        error_code = SPI_Configuration_Set( board_id, &cfg );

```

```

48: if ( SUCCESS != error_code ) return error_code;
49: printf( "OK\n" );
50: }
51: return SUCCESS;
52: }
```

3.2.1.92 CMD__SPI_Config_Chip_Select_Route Function

C++

```
static int CMD__SPI_Config_Chip_Select_Route(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__SPI_Config_Chip_Select_Route( int board_id, int argc, char * argv[ ] )
2: {
3:     int     error_code;
4:
5:     struct    spi_cfg cfg;
6:
7:
8:     error_code = SPI_Configuration_Get( board_id, &cfg );
9:     if ( SUCCESS != error_code ) return error_code;
10:
11:    if ( argc < 1 )
12:    {
13:        printf( "SPI CS ROUTE: %s\n", cfg.chip_select_route_ch1 ? "1" : "0" );
14:    }
15:    else
16:    {
17:        cfg.chip_select_route_ch1 = String_To_Bool( argv[0] );
18:
19:        error_code = SPI_Configuration_Set( board_id, &cfg );
20:        if ( SUCCESS != error_code ) return error_code;
21:        printf( "OK\n" );
22:    }
23:    return SUCCESS;
24: }
```

3.2.1.93 CMD__SPI_Config_Clock_Hz Function

C++

```
static int CMD__SPI_Config_Clock_Hz(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__SPI_Config_Clock_Hz( int board_id, int argc, char * argv[] )
2: {
3:     int     error_code;
4:     double   clock_request_hz;
5:     double   clock_actual_hz;
6:     double   error;
7:     uint16_t hci;
8:     struct    spi_cfg cfg;
9:
10:
11:    error_code = SPI_Configuration_Get( board_id, &cfg );
12:    if ( SUCCESS != error_code ) return error_code;
13:
14:    if ( argc < 1 )
15:    {
16:        printf( "SPI CLK: %f hz\n", cfg.clock_hz );
17:    }
18:    else
19:    {
20:        clock_request_hz = atof( argv[0] );
21:        error_code = SPI_Calculate_Clock( clock_request_hz, &clock_actual_hz, &error, &hci );
22:        if ( SUCCESS != error_code ) return error_code;
23:
24:
25:        cfg.clock_hz = clock_actual_hz;
26:        error_code   = SPI_Configuration_Set( board_id, &cfg );
27:        if ( SUCCESS != error_code ) return error_code;
28:        printf( "OK\n" );
29:    }
30:    return SUCCESS;
31: }
```

3.2.1.94 CMD__SPI_Config_End_Cycle_Delay_Sec Function

C++

```
static int CMD__SPI_Config_End_Cycle_Delay_Sec(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__SPI_Config_End_Cycle_Delay_Sec( int board_id, int argc, char * argv[] )
2: {
3:     int     error_code;
4:     double   request_sec;
5:     double   actual_sec;
6:     double   error;
7:     uint8_t  ecd;
8:     struct    spi_cfg cfg;
9:
10:
11:    error_code = SPI_Configuration_Get( board_id, &cfg );
12:    if ( SUCCESS != error_code ) return error_code;
13:
14:    if ( argc < 1 )
15:    {
16:        printf( "SPI ECD: %g sec\n", ( cfg.end_delay_ns * 1.0e-9 ) );
17:    }
18:    else
19:    {
20:        request_sec = atof( argv[0] );
21:        error_code = SPI_Calculate_End_Cycle_Delay( SPI_Calculate_Half_Clock_Interval_Sec(
22:            cfg.half_clock_interval ),
23:            request_sec,
24:            &actual_sec,
25:            &error,
26:            &ecd
27:        );
28:        if ( SUCCESS != error_code ) return error_code;
29:        cfg.end_delay_ns = actual_sec * 1.0e9;
30:
31:        error_code = SPI_Configuration_Set( board_id, &cfg );
32:        if ( SUCCESS != error_code ) return error_code;
33:        printf( "OK\n" );
34:    }
35:    return SUCCESS;
}

```

3.2.1.95 CMD__SPI_Config_Get Function

C++

```
static int CMD__SPI_Config_Get(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__SPI_Config_Get( int board_id, int argc, char * argv[] )
2: {
3:     int error_code;
4:     struct spi_cfg cfg;
5:     (void) argc;
6:     (void) argv;
7:
8:     error_code = SPI_Configuration_Get( board_id, &cfg );
9:     if ( SUCCESS != error_code ) return error_code;
10:
11:    error_code = SPI_Report_Configuration_Text( &cfg, stdout );
12:    if ( SUCCESS != error_code ) return error_code;
13:
14:    printf( "\n" );
15:    return SUCCESS;
16: }
```

3.2.1.96 CMD__SPI_Config_Mode Function

CPOL CPHA MODE 0 0 0 1 0 1 2 1 0 3 1 1

* *

- @brief
- @return A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

C++

```
static int CMD__SPI_Config_Mode(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Body Source

```

1: static int CMD__SPI_Config_Mode( int board_id, int argc, char * argv[] )
2: {
3:     int error_code;
4:     int mode;
5:     struct spi_cfg cfg;
6:
7:
```

```

8: error_code = SPI_Configuration_Get( board_id, &cfg );
9: if ( SUCCESS != error_code ) return error_code;
10:
11: if ( argc < 1 )
12: {
13:     if ( (false == cfg.sclk_polarity) && (false == cfg.sclk_phase) ) mode = 0;
14:     else if ( (false == cfg.sclk_polarity) && (true == cfg.sclk_phase) ) mode = 1;
15:     else if ( (true == cfg.sclk_polarity) && (false == cfg.sclk_phase) ) mode = 2;
16:     else if ( (true == cfg.sclk_polarity) && (true == cfg.sclk_phase) ) mode = 3;
17:     printf( "SPI MODE: %d\n", mode );
18: }
19: else
20: {
21:     mode = (int) strtol( argv[0], NULL, 0 );
22:     switch ( mode )
23:     {
24:         case 0: cfg.sclk_polarity = false; cfg.sclk_phase = false; break;
25:         case 1: cfg.sclk_polarity = false; cfg.sclk_phase = true; break;
26:         case 2: cfg.sclk_polarity = true; cfg.sclk_phase = false; break;
27:         case 3: cfg.sclk_polarity = true; cfg.sclk_phase = true; break;
28:     }
29:
30:     error_code = SPI_Configuration_Set( ID_IDI48, &cfg );
31:     if ( SUCCESS != error_code ) return error_code;
32:     printf( "OK\n" );
33: }
34: return SUCCESS;
35: }
```

3.2.1.97 CMD__SPI_Config_SDI_Polarity Function

C++

```
static int CMD__SPI_Config_SDI_Polarity(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__SPI_Config_SDI_Polarity( int board_id, int argc, char * argv[] )
2: {
3:     int error_code;
4:
5:     struct spi_cfg cfg;
6:
7:
8:     error_code = SPI_Configuration_Get( board_id, &cfg );
9:     if ( SUCCESS != error_code ) return error_code;
10:
11:    if ( argc < 1 )
```

```

12: {
13:     printf( "SPI SDI POLARITY: %s\n", cfg.sdi_polarity ? "true" : "false" );
14: }
15: else
16: {
17:     cfg.sdi_polarity = String_To_Bool( argv[0] );
18:
19:     error_code = SPI_Configuration_Set( board_id, &cfg );
20:     if ( SUCCESS != error_code ) return error_code;
21:     printf( "OK\n" );
22: }
23: return SUCCESS;
24: }
```

3.2.1.98 CMD__SPI_Config_SDIO_Wrap Function

C++

```
static int CMD__SPI_Config_SDIO_Wrap(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__SPI_Config_SDIO_Wrap( int board_id, int argc, char * argv[] )
2: {
3:     int error_code;
4:
5:     struct spi_cfg cfg;
6:
7:
8:     error_code = SPI_Configuration_Get( board_id, &cfg );
9:     if ( SUCCESS != error_code ) return error_code;
10:
11:    if ( argc < 1 )
12:    {
13:        printf( "SPI wrap: %s\n", cfg.sdio_wrap ? "true" : "false" );
14:    }
15: else
16: {
17:     cfg.sdio_wrap = String_To_Bool( argv[0] );
18:
19:     error_code = SPI_Configuration_Set( board_id, &cfg );
20:     if ( SUCCESS != error_code ) return error_code;
21:     printf( "OK\n" );
22: }
23: return SUCCESS;
24: }
```

3.2.1.99 CMD__SPI_Config_SDO_Polarity Function

C++

```
static int CMD__SPI_Config_SDO_Polarity(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```
1: static int CMD__SPI_Config_SDO_Polarity( int board_id, int argc, char * argv[] )
2: {
3:     int     error_code;
4:
5:     struct    spi_cfg cfg;
6:
7:
8:     error_code = SPI_Configuration_Get( board_id, &cfg );
9:     if ( SUCCESS != error_code ) return error_code;
10:
11:    if ( argc < 1 )
12:    {
13:        printf( "SPI SDO POLARITY: %s\n", cfg.sdo_polarity ? "true" : "false" );
14:    }
15:    else
16:    {
17:        cfg.sdo_polarity = String_To_Bool( argv[0] );
18:
19:        error_code = SPI_Configuration_Set( board_id, &cfg );
20:        if ( SUCCESS != error_code ) return error_code;
21:        printf( "OK\n" );
22:    }
23:    return SUCCESS;
24: }
```

3.2.1.100 CMD__SPI_Data Function

C++

```
static int CMD__SPI_Data(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Todo

add a command to allow different object sizes to be transmitted. That would better demonstrate the buffer (see page 325) mode in the SPI hardware. Currently, if csb=buffer (see page 325) then only a byte per spi fifo buffer (see page 325) will be transmitted. All bytes will be transmitted, each byte will be chip-select wrapped and overall SPI transactions will be slow due to software and bus timing.

Body Source

```

1: static int CMD__SPI_Data( int board_id, int argc, char * argv[ ] )
2: {
3:     int         error_code;
4:     size_t      index;
5:     size_t      transfer_count;
6:     size_t      lines;
7:     uint8_t *   bp;
8:
9:     struct board_dataset * dataset;
10:    dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
11:
12:
13:
14:    if ( argc < 1 ) return -EC_PARAMETER;
15:
16:    CMD__SPI_Data_Interpreter( 0,
17:                                argc,
18:                                argv,
19:                                SPI_BLOCK_SIZE,
20:                                &transfer_count,
21:                                dataset->spi.tx_buffer
22:                                );
23:
24:    error_code = SPI_Data_Write_Read( board_id,
25:                                      sizeof( uint8_t ),
26:                                      transfer_count,
27:                                      dataset->spi.tx_buffer,
28:                                      transfer_count,
29:                                      dataset->spi.rx_buffer
30:                                      );
31:    if ( SUCCESS != error_code ) return error_code;
32:
33:    lines = transfer_count / HEX_DUMP_BYTES_PER_LINE;
34:
35:    for ( index = 0; index <= lines; index++ )
36:    {
37:        bp = &(dataset->spi.rx_buffer[index * HEX_DUMP_BYTES_PER_LINE]);
38:        if ( transfer_count < HEX_DUMP_BYTES_PER_LINE )
39:        {
40:            Hex_Dump_Line( 0, transfer_count, bp, stdout );
41:        }
42:        else
43:        {
44:            Hex_Dump_Line( 0, HEX_DUMP_BYTES_PER_LINE, bp, stdout );

```

```

45:     transfer_count = transfer_count - HEX_DUMP_BYTES_PER_LINE;
46: }
47: }
48: return SUCCESS;
49: }

```

3.2.1.101 CMD__SPI_Data_Interpreter Function

C++

```
static int CMD__SPI_Data_Interpreter(int arg_start, int argc, char * argv[], size_t tx_size,
size_t * tx_count, uint8_t * tx_buffer);
```

File

idi.c (see page 373)

Parameters

Parameters	Description
int arg_start	[in] starting argument
size_t tx_size	[in] max size of out buffer in bytes
size_t * tx_count	[out] actual size
uint8_t * tx_buffer	[out] output buffer

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__SPI_Data_Interpreter( int arg_start,
2:                                     int argc,
3:                                     char * argv[],
4:                                     size_t tx_size,
5:                                     size_t * tx_count,
6:                                     uint8_t * tx_buffer
7:                                     )
8: {
9:     size_t arg_qty;
10:    size_t count;
11:    int index_arg;
12:    int index_byte;
13:    uint8_t data_temp;
14:    char * endptr;
15:    char * sp;
16:
17:    arg_qty = (size_t) ( argc - arg_start );
18:
19:    sp = NULL;
20:    index_arg = arg_start;
21:    index_byte = 0;
22:    count = arg_qty;
23:    while ( ( count != 0 ) && ( index_byte < tx_size ) )

```

```

24: {
25:     data_temp = (uint8_t) strtol( argv[index_arg], &endptr, 0 );
26:     if ( endptr == argv[index_arg] )
27:     {
28:         sp = &(argv[index_arg][0]);
29:         while ( ( '\0' != *sp ) && ( index_byte < tx_size ) )
30:         {
31:             tx_buffer[index_byte] = (uint8_t)( *sp );
32:             sp++; index_byte++;
33:         }
34:         if ( '\0' == *sp ) sp = NULL;
35:     }
36:     else
37:     {
38:         tx_buffer[index_byte] = data_temp;
39:         index_byte++;
40:         sp = NULL;
41:     }
42:     count--; index_arg++;
43: }
44:
45: if ( count > 0 )
46: {
47: #if defined( __MSDOS__ )
48:     printf( "Warning: ignored %d arguments", count );
49: #else
50:     printf( "Warning: ignored %lu arguments", count );
51: #endif
52:     if ( NULL != sp )
53:     {
54:         printf( ", and portion of string argument" );
55:     }
56:     printf( "\n" );
57: }
58:
59: *tx_count = index_byte;
60: return SUCCESS;
61: }
```

3.2.1.102 CMD__SPI_FIFO Function

C++

```
static int CMD__SPI_FIFO(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```
1: static int CMD__SPI_FIFO( int board_id, int argc, char * argv[ ] )
```

```
2: {
3:     int      error_code;
4:     int      index;
5:     size_t   count;
6:     SPI_CSB_ENUM csb;
7:     int      read_count;
8:
9:     uint8_t    tx_buffer[SPI_FIFO_SIZE];
10:    uint8_t   rx_buffer[SPI_FIFO_SIZE];
11:
12:    struct board_dataset * dataset;
13:    dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
14:
15:    read_count = 0;
16:    if ( argc < 1 )
17:    {
18:        read_count = SPI_FIFO_SIZE;
19:    }
20:    else
21:    {
22:        if ( 0 == strcasecmp( "rx", argv[0] ) )
23:        {
24:            if ( argc > 1 )
25:            {
26:                if ( 0 == strcasecmp( "all", argv[1] ) ) read_count = SPI_FIFO_SIZE;
27:                else read_count = (int) strtol( argv[1], NULL, 0 );
28:            }
29:        }
30:        else if ( 0 == strcasecmp( "tx", argv[0] ) )
31:        {
32:            CMD__SPI_Data_Interpreter( 1,
33:                argc,
34:                argv,
35:                SPI_FIFO_SIZE,
36:                &count,
37:                tx_buffer
38:            );
39:
40:            error_code = SPI_FIFO_Write( board_id, (void *) tx_buffer, sizeof( uint8_t ), count,
NULL );
41:            if ( error_code < 0 ) return error_code;
42:
43:            printf( "OK, wrote %d objects or %d bytes\n", error_code, error_code * ((int) sizeof(
uint8_t ) ) );
44:            return SUCCESS;
45:        }
46:        else if ( 0 == strcasecmp( "commit", argv[0] ) )
47:        {
48:            if ( argc > 1 )
49:            {
50:                if ( String_To_Bool( argv[1] ) ) SPI_Commit( board_id, 0xFF );
51:                else                           SPI_Commit( board_id, 0x00 );
52:
53:                printf( "OK\n" );
54:                return SUCCESS;
55:            }
56:        else
57:        {
58:            uint8_t commit_status;
59:            SPI_Commit_Get( board_id, &commit_status );
60:            printf( "commit status = %s\n", ( 0 == commit_status ) ? "false" : "true" );
61:            return SUCCESS;
62:        }
63:
```

```
64:    }
65: }
66:
67:
68:
69:
70: error_code = SPI_Configuration_Chip_Select_Behavior_Get( board_id, &csb );
71: if ( SUCCESS != error_code ) return error_code;
72:
73: if ( SPI_Status_Write_FIFO_Is_Not_Empty( board_id ) )
74: {
75:   if ( CSB_SOFTWARE != csb ) SPI_Commit( board_id, 0xFF );
76: }
77:
78: while ( SPI_Status_Write_FIFO_Is_Not_Empty( board_id ) )
79: {
80:   error_code = SUCCESS;
81:   if ( dataset->quit_application ) error_code = -EC_APP_TERMINATE_CTRLC;
82:   if ( Character_Get( NULL ) ) error_code = -EC_APP_TERMINATE;
83:   if ( SUCCESS != error_code )
84:   {
85:     printf( "IDI_CMD_SPI_FIFO: waiting for tx empty\n" );
86:     return -EC_APP_TERMINATE;
87:   }
88: }
89:
90: if ( read_count > 0 )
91: {
92:   int lines;
93:   uint8_t * bp;
94:
95:   if ( SPI_Status_Read_FIFO_Is_Not_Empty( board_id ) )
96:   {
97:     error_code = SPI_FIFO_Read( board_id, (void *) rx_buffer, sizeof( uint8_t ),
read_count, NULL );
98:     if ( error_code < 0 ) return error_code;
99:
100:    read_count = error_code * sizeof( uint8_t );
101:    lines = read_count / HEX_DUMP_BYTES_PER_LINE;
102:
103:    for ( index = 0; index <= lines; index++ )
104:    {
105:      bp = &(rx_buffer[index * HEX_DUMP_BYTES_PER_LINE]);
106:      if ( read_count < HEX_DUMP_BYTES_PER_LINE )
107:      {
108:        Hex_Dump_Line( 0, read_count, bp, stdout );
109:      }
110:      else
111:      {
112:        Hex_Dump_Line( 0, HEX_DUMP_BYTES_PER_LINE, bp, stdout );
113:        read_count = read_count - HEX_DUMP_BYTES_PER_LINE;
114:      }
115:    }
116:  }
117: else
118: {
119:   printf( "FIFO Empty\n" );
120: }
121: }
122: return SUCCESS;
123:
124:
125:
126:
```

```
127: }
```

3.2.1.103 CMD__SPI_ID Function

C++

```
static int CMD__SPI_ID(int board_id, int argc, char * argv[ ]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```
1: static int CMD__SPI_ID( int board_id, int argc, char * argv[ ] )
2: {
3:     uint16_t id;
4:     (void) argc;
5:     (void) argv;
6:
7:     SPI_ID_Get( board_id, &id );
8:     printf( "SPI ID: 0x%04X\n", id );
9:     return SUCCESS;
10: }
```

3.2.1.104 CMD__SPI_Status Function

C++

```
static int CMD__SPI_Status(int board_id, int argc, char * argv[ ]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```
1: static int CMD__SPI_Status( int board_id, int argc, char * argv[ ] )
2: {
3:     int error_code;
```

```

4: int index;
5: struct spi_status status;
6:
7: if ( argc < 1 )
8: {
9:     error_code = SPI_Status_Read( board_id, &status );
10:    if ( SUCCESS != error_code ) return error_code;
11:    SPI_Report_Status_Text( &status, stdout );
12: }
13: else
14: {
15:     for ( index = 0; index < argc; index++ )
16:     {
17:         if ( 0 == strcmpi( "rx", argv[index] ) )
18:         {
19:             error_code = SPI_Status_Read( board_id, &status );
20:             if ( SUCCESS != error_code ) return error_code;
21:             SPI_Report_Status_Text( &status, stdout );
22:         }
23:         else if ( 0 == strcmpi( "tx", argv[index] ) )
24:         {
25:             error_code = SPI_Status_Write( board_id, &status );
26:             if ( SUCCESS != error_code ) return error_code;
27:             SPI_Report_Status_Text( &status, stdout );
28:         }
29:     }
30: }
31: return SUCCESS;
32: }
```

3.2.1.105 CMD__Trace_File Function

C++

```
static int CMD__Trace_File(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__Trace_File( int board_id, int argc, char * argv[] )
2: {
3:     int error_code;
4:     size_t index;
5:     BOOL not_done;
6:     struct board_dataset * dataset;
7:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
8:
9:     error_code = SUCCESS;
10:    if ( argc < 1 )
```

```

11:  {
12:    printf( "file_name      = " );
13:    if ( 0 == strlen( dataset->trace.filename ) ) printf( "%s\n", IO_TRACE_DUMP_FILE_NAME );
14:    else                                         printf( "%s\n", dataset->trace.filename );
15:  }
16:  else
17:  {
18:    index      = 0;
19:    not_done   = true;
20:    do
21:    {
22:      dataset->trace.filename[index] = argv[0][index];
23:      if ( '\0' == argv[0][index] )
24:      {
25:        not_done = false;
26:      }
27:      else if ( index >= ( IO_TRACE_FILE_NAME_SIZE - 2 ) )
28:      {
29:        not_done = false;
30:        index++;
31:        dataset->trace.filename[index] = '\0';
32:      }
33:      else
34:      {
35:        index++;
36:      }
37:    } while ( true == not_done );
38:  }
39:  return error_code;
40: }
```

3.2.1.106 CMD__Trace_Start Function

C++

```
static int CMD__Trace_Start(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__Trace_Start( int board_id, int argc, char * argv[] )
2: {
3:   int      error_code;
4:   struct board_dataset * dataset;
5:   dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
6:
7:   error_code = SUCCESS;
8:   if ( argc < 1 )
9:   {
```

```

10: printf( "start      = " );
11: printf( "%s\n", global_io_trace_start_name[dataset->trace.start] );
12:
13: }
14: else
15: {
16:     int index = 0;
17:     error_code = -EC_PARAMETER;
18:     while ( NULL != global_io_trace_start_name[index] )
19:     {
20:         if ( 0 == strcasecmp( global_io_trace_start_name[index], argv[0] ) )
21:         {
22:             dataset->trace.start = (enum IO_TRACE_START_ENUM) index;
23:             error_code = SUCCESS;
24:             break;
25:         }
26:         index++;
27:     }
28: }
29: return error_code;
30: }
```

3.2.1.107 CMD__Trace_Stop Function

C++

```
static int CMD__Trace_Stop(int board_id, int argc, char * argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int CMD__Trace_Stop( int board_id, int argc, char * argv[] )
2: {
3:     int error_code;
4:     struct board_dataset * dataset;
5:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
6:
7:     error_code = SUCCESS;
8:     if ( argc < 1 )
9:     {
10:         printf( "stop      = " );
11:         printf( "%s\n", global_io_trace_stop_name[dataset->trace.stop] );
12:
13:     }
14:     else
15:     {
16:         int index = 0;
17:         error_code = -EC_PARAMETER;
```

```

26:     while ( NULL != global_io_trace_stop_name[index] )
27:     {
28:         if ( 0 == strcasecmp( global_io_trace_stop_name[index], argv[0] ) )
29:         {
30:             dataset->trace.stop = (enum IO_TRACE_STOP_ENUM) index;
31:             error_code = SUCCESS;
32:             break;
33:         }
34:         index++;
35:     }
36:
37: }
38: return error_code;
39: }
```

3.2.1.108 Command_Line_Analog_Stick Function

C++

```
int Command_Line_Analog_Stick(int board_id, int argc, char* argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: int Command_Line_Analog_Stick( int board_id, int argc, char* argv[ ] )
2: {
3:     int error_code;
4:     int index;
5:     int argc_new;
6:     char ** argv_new;
7:
8:     error_code = -EC_SYNTAX;
9:
10:    if ( argc < 1 ) return -EC_NOT_FOUND;
11:
12:    index = 0;
13:    while ( NULL != cmd_as[index].cmd_fnc )
14:    {
15:        if ( 0 == strcasecmp( cmd_as[index].name, argv[0] ) )
16:        {
17:            argv_new = &(argv[1]);
18:            argc_new = argc - 1;
19:            if ( 0 == argc_new ) argv_new = NULL;
20:
21:            {
22:                struct board_dataset * dataset;
23:                dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
24:                if ( IO_TRACE_START_AS == dataset->trace.start )
25:                {
```

```

26:     dataset->trace.active = true;
27: }
28: }
29:
30: error_code = (* cmd_as[index].cmd_fnc )( board_id, argc_new, argv_new );
31: break;
32: }
33: index++;
34: }
35: return error_code;
36: }
```

3.2.1.109 Command_Line_Dump Function

C++

```
int Command_Line_Dump(int board_id, int argc, char* argv[ ]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: int Command_Line_Dump( int board_id, int argc, char* argv[ ] )
2: {
3:     int     error_code;
4:     FILE * fd_out;
5:
6:
7:
8:     if ( argc > 0 )
9:     {
10:        fd_out = fopen( argv[0], "w" );
11:        if ( NULL == fd_out ) fd_out = stdout;
12:    }
13:    else
14:    {
15:        fd_out = stdout;
16:    }
17:    error_code = Register_Report_CSV( board_id, fd_out );
18:
19:    if ( (argc > 0) && (NULL != fd_out) && (stdout != fd_out) )
20:    {
21:        fclose( fd_out );
22:    }
23:    return error_code;
24: }
```

3.2.1.110 Command_Line_FPGA Function

C++

```
int Command_Line_FPGA(int board_id, int argc, char* argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```
1: int Command_Line_FPGA( int board_id, int argc, char* argv[ ] )
2: {
3:     int     error_code;
4:     FILE * out;
5:     uint32_t date_svn;
6:     uint32_t date_compilation;
7:     (void) argc;
8:     (void) argv;
9:
10:    error_code = FPGA_Date_Time_Information( board_id, &date_svn, &date_compilation );
11:    if ( error_code < SUCCESS ) goto Command_Line_FPGA_Exit;
12:
13:    out = stdout;
14:
15:    fprintf( out, "FPGA: " );
16:
17:    if ( 0 == date_svn )
18:    {
19:        fprintf( out, "SVN = unavailable" );
20:    }
21:    else
22:    {
23: #if defined( __MSDOS__ )
24:        fprintf( out, "SVN =      %08lu", date_svn );
25: #else
26:        fprintf( out, "SVN =      %08u", date_svn );
27: #endif
28:    }
29:
30:    if ( 0 == date_compilation )
31:    {
32:        fprintf( out, ", Compilation = unavailable\n" );
33:    }
34:    else
35:    {
36: #if defined( __MSDOS__ )
37:        fprintf( out, ", Compilation = %08lu\n", date_compilation );
38: #else
39:        fprintf( out, ", Compilation = %08u\n", date_compilation );
```

```

40: #endif
41: }
42:
43: Command_Line_FPGA_Exit:
44:     return error_code;
45: }
```

3.2.1.111 Command_Line_FRAM Function

C++

```
int Command_Line_FRAM(int board_id, int argc, char* argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: int Command_Line_FRAM( int board_id, int argc, char* argv[ ] )
2: {
3:     int     error_code;
4:     int     index;
5:     int     argc_new;
6:     char ** argv_new;
7:
8:     error_code = -EC_SYNTAX;
9:
10:    if ( argc < 1 ) return -EC_NOT_FOUND;
11:
12:    index = 0;
13:    while ( NULL != cmd_fram[index].cmd_fnc )
14:    {
15:        if ( 0 == strcasecmp( cmd_fram[index].name, argv[0] ) )
16:        {
17:            argv_new = &(argv[1]);
18:            argc_new = argc - 1;
19:            if ( 0 == argc_new ) argv_new = NULL;
20:
21:        }
22:        struct board_dataset * dataset;
23:        dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
24:        if ( IO_TRACE_START_FRAM == dataset->trace.start )
25:        {
26:            dataset->trace.active = true;
27:        }
28:    }
29:
30:    error_code = (* cmd_fram[index].cmd_fnc )( board_id, argc_new, argv_new );
31:    break;
32: }
33: index++;
```

```
34: }
35: return error_code;
36: }
```

3.2.1.112 Command_Line_Help Function

C++

```
int Command_Line_Help(int board_id, int argc, char* argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```
1: int Command_Line_Help( int board_id, int argc, char* argv[ ] )
2: {
3:     FILE * out;
4:     FILE * fd = NULL;
5:     BOOL skip_pause = false;
6:     (void) board_id;
7:
8:     out = stdout;
9:     if ( argc > 0 )
10:    {
11:        fd = fopen( argv[0], "w" );
12:        if ( NULL != fd )
13:        {
14:            out = fd; skip_pause = true;
15:        }
16:    }
17:    Help( skip_pause, out );
18:    if ( NULL != fd ) fclose( fd );
19:
20:    return SUCCESS;
21: }
```

3.2.1.113 Command_Line_IAI16 Function

C++

```
int Command_Line_IAI16(int board_id, int argc, char* argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: int Command_Line_IAI16( int board_id, int argc, char* argv[] )
2: {
3:     int    error_code;
4:     int    index;
5:     int    argc_new;
6:     char ** argv_new;
7:     char *    endptr;
8:
9:
10:    if ( argc < 1 ) return -EC_NOT_FOUND;
11:
12:    if ( ID_IDO48 != board_id ) return -EC_BOARD_TYPE;
13:
14:    error_code = -EC_SYNTAX;
15:
16:
17:    strtol( argv[0], &endptr, 0 );
18:    if ( argv[0] != endptr )
19:    {
20:        error_code = (* cmd_iai16[0].cmd_fnc )( board_id, argc, argv );
21:    }
22:    else
23:    {
24:        index = 0;
25:        while ( NULL != cmd_iai16[index].cmd_fnc )
26:        {
27:            if ( 0 == strcasecmp( cmd_iai16[index].name, argv[0] ) )
28:            {
29:                argv_new = &(argv[1]);
30:                argc_new = argc - 1;
31:                if ( 0 == argc_new ) argv_new = NULL;
32:                error_code = (* cmd_iai16[index].cmd_fnc )( board_id, argc_new, argv_new );
33:                break;
34:            }
35:            index++;
36:        }
37:    }
38:    return error_code;
39: }
```

3.2.1.114 Command_Line_IDI48 Function

C++

```
int Command_Line_IDI48(int board_id, int argc, char* argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```
1: int Command_Line_IDI48( int board_id, int argc, char* argv[ ] )
2: {
3:     int     error_code;
4:     int     index;
5:     int     argc_new;
6:     char ** argv_new;
7:     char *    endptr;
8:
9:
10:    if ( argc < 1 ) return -EC_NOT_FOUND;
11:
12:    if ( ID_IDI48 != board_id ) return -EC_BOARD_TYPE;
13:
14:    error_code = -EC_SYNTAX;
15:
16:
17:    strtol( argv[0], &endptr, 0 );
18:    if ( argv[0] != endptr )
19:    {
20:        error_code = (* cmd_idi48[0].cmd_fnc )( board_id, argc, argv );
21:    }
22:    else
23:    {
24:        index = 0;
25:        while ( NULL != cmd_idi48[index].cmd_fnc )
26:        {
27:            if ( 0 == strcasecmp( cmd_idi48[index].name, argv[0] ) )
28:            {
29:                argv_new = &(argv[1]);
30:                argc_new = argc - 1;
31:                if ( 0 == argc_new ) argv_new = NULL;
32:                error_code = (* cmd_idi48[index].cmd_fnc )( board_id, argc_new, argv_new );
33:                break;
34:            }
35:            index++;
36:        }
37:    }
38:    return error_code;
39: }
```

3.2.1.115 Command_Line_IDO48 Function

C++

```
int Command_Line_IDO48(int board_id, int argc, char* argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: int Command_Line_IDO48( int board_id, int argc, char* argv[] )
2: {
3:     int    error_code;
4:     int    index;
5:     int    argc_new;
6:     char ** argv_new;
7:     char *    endptr;
8:
9:
10:    if ( argc < 1 ) return -EC_NOT_FOUND;
11:
12:    if ( ID_IDO48 != board_id ) return -EC_BOARD_TYPE;
13:
14:    error_code = -EC_SYNTAX;
15:
16:
17:    strtol( argv[0], &endptr, 0 );
18:    if ( argv[0] != endptr )
19:    {
20:        error_code = (* cmd_ido48[0].cmd_fnc )( board_id, argc, argv );
21:    }
22:    else
23:    {
24:        index = 0;
25:        while ( NULL != cmd_ido48[index].cmd_fnc )
26:        {
27:            if ( 0 == strcasecmp( cmd_ido48[index].name, argv[0] ) )
28:            {
29:                argv_new = &(argv[1]);
30:                argc_new = argc - 1;
31:                if ( 1 == argc_new ) argv_new = NULL;
32:                error_code = (* cmd_ido48[index].cmd_fnc )( board_id, argc_new, argv_new );
33:                break;
34:            }
35:            index++;
36:        }
37:    }
38:    return error_code;
39: }
```

3.2.1.116 Command_Line_IO_Read Function

C++

```
int Command_Line_IO_Read(int board_id, int argc, char* argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief used to read one or more values from registers sequentially.

b [prefix commands] [x] [c]

Body Source

```
1: int Command_Line_IO_Read( int board_id, int argc, char* argv[] )  
2: {  
3:     size_t index;  
4:     BOOL count_capture;  
5:     BOOL address_captured;  
6:     BOOL auto_index;  
7:     int     error_code;  
8:     int     address;  
9:     int     count;  
10:  
11:    uint8_t value;  
12:    (void) board_id;  
13:  
14:    if ( argc < 1 ) return -EC_NOT_FOUND;  
15:  
16:    count      = 1;  
17:    error_code     = SUCCESS;  
18:    count_capture   = false;  
19:    address_captured = false;  
20:    auto_index     = true;  
21:    for ( index = 0; index < argc; index++ )  
22:    {  
23:        if ( ( 'c' == argv[index][0] ) || ( 'C' == argv[index][0] ) )  
24:        {  
25:            count_capture = true;  
26:        }  
27:        else if ( ( 'x' == argv[index][0] ) || ( 'X' == argv[index][0] ) )  
28:        {  
29:            auto_index = !auto_index;  
30:        }  
31:        else  
32:        {  
33:            if ( true == count_capture )  
34:            {  
35:                count = (int) strtol( argv[index], NULL, 0 );  
36:                count_capture = false;  
37:            }  
38:            else  
39:            {  
40:                if ( false == address_captured )  
41:                {  
42:                    address = (int) strtol( argv[index], NULL, 0 );  
43:                    address_captured = true;  
44:                    printf( "0x%04X:", address );  
45:                }  
46:                else  
47:                {  
48:  
49:                }  
50:            }  
51:        }  
52:    }  
53: }
```

```

51:     }
52:   }
53: }
54:
55: for ( index = 0; index < count; index++ )
56: {
57:   error_code = IO_Read_U8_Port( NULL, __LINE__, address, &value );
58:   if ( SUCCESS != error_code ) return error_code;
59:   printf( " 0x%02X", value );
60:   if ( true == auto_index ) address++;
61: }
62:
63: printf( "\n" );
64: return error_code;
65: }
```

3.2.1.117 Command_Line_IO_Write Function

C++

```
int Command_Line_IO_Write(int board_id, int argc, char* argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Used to write to one or more registers in sequence. Can write values to the same register by using the 'x' to toggle auto-indexing from initially on to the off position.

Body Source

```

1: int Command_Line_IO_Write( int board_id, int argc, char* argv[] )
2: {
3:   size_t index;
4:   BOOL address_captured;
5:   int error_code;
6:   int address;
7:   uint8_t value;
8:   BOOL auto_index;
9:   (void) board_id;
10:
11: if ( argc < 2 ) return -EC_NOT_FOUND;
12:
13: address_captured = false;
14: auto_index = true;
15: for ( index = 0; index < argc; index++ )
16: {
17:   if ( ( 'x' == argv[index][0] ) || ( 'X' == argv[index][0] ) )
18:   {
19:     auto_index = !auto_index;
20:   }
21:   else
22:   {
23:     if ( false == address_captured )
```

```

24:     {
25:         address = (int) strtol( argv[index], NULL, 0 );
26:         address_captured = true;
27:
28:     }
29:     else
30:     {
31:         value   = (uint8_t) strtol( argv[index], NULL, 0 );
32:         error_code = IO_Write_U8_Port( NULL, __LINE__, address, value );
33:         if ( SUCCESS != error_code ) return error_code;
34:         if ( true == auto_index ) address++;
35:     }
36: }
37: }
38: printf( "OK\n" );
39: return error_code;
40: }
```

3.2.1.118 Command_Line_Loop_Count Function

C++

```
static int Command_Line_Loop_Count(int argc, char* argv[]);
```

File

idi.c (see page 373)

Parameters

Parameters	Description
int argc	argument index in this case
char* argv[]	always null in this case

Description

brief

Body Source

```

1: static int Command_Line_Loop_Count( int    argc,
2:                                     char* argv[ ]
3:                                     )
4: {
5:     int error_code;
6:
7:     if ( argc > 0 )
8:     {
9:         if ( argv[0][0] >= '0' && argv[0][0] <= '9' )
10:        {
11:            global_loop_count      = (int) strtol( argv[0], NULL, 0 );
12:            global_loop_count_counter = global_loop_count;
13:            error_code = 1;
14:        }
15:     else
16:     {
17:         error_code = -EC_SYNTAX;
18:     }
19: }
```

```

20: else
21: {
22:     error_code = -EC_SYNTAX;
23: }
24: return error_code;
25: }
```

3.2.1.119 Command_Line_Loop_Delay Function

C++

```
static int Command_Line_Loop_Delay(int argc, char* argv[ ]);
```

File

idi.c (see page 373)

Parameters

Parameters	Description
int argc	argument index in this case
char* argv[]	always null in this case

Description

brief

Body Source

```

1: static int Command_Line_Loop_Delay( int    argc,
2:                                     char* argv[ ]
3:                                     )
4: {
5:     int error_code;
6:
7:     if ( argc > 0 )
8:     {
9:         if ( ( argv[0][0] >= '0' ) && ( argv[0][0] <= '9' ) )
10:        {
11:            global_loop_delay_ms = (int) strtol( argv[0], NULL, 0 );
12:            error_code = 1;
13:        }
14:    }
15:    else
16:    {
17:        error_code = -EC_SYNTAX;
18:    }
19: }
20: else
21: {
22:     error_code = -EC_SYNTAX;
23: }
24: return error_code;
}
```

3.2.1.120 Command_Line_Loop_Space Function

C++

```
static int Command_Line_Loop_Space(int argc, char* argv[ ]);
```

File

idi.c (see page 373)

Parameters

Parameters	Description
int argc	argument index in this case
char* argv[]	always null in this case

Description

brief

Body Source

```
1: static int Command_Line_Loop_Space( int    argc,
2:                                     char* argv[ ]
3:                                     )
4: {
5:   (void) argc;
6:   (void) argv;
7:
8:   global_loop_space = true;
9:   return SUCCESS;
10: }
```

3.2.1.121 Command_Line_Main Function

C++

```
int Command_Line_Main(int board_id, int argc, char* argv[ ]);
```

File

idi.c (see page 373)

Returns

SUCCESS (0) if no errors encountered, otherwise errors are reported as a negative value.

Description

brief Processes and dispatches the top level of the command and passes the remaining string list onto specialized functions to further process arguments. If no command is specified then a help output is produced.

param[in] argc number of arguments including the executable file name param[in] argv list of string arguments lex'd from the command line

Body Source

```

1: int Command_Line_Main( int board_id, int argc, char* argv[ ] )
2: {
3:     int error_code;
4:     int index;
5:     BOOL not_found;
6:     int argc_new;
7:     char ** argv_new;
8:
9:
10:    error_code = -EC_SYNTAX;
11:
12:    if ( argc < 1 ) return -EC_NOT_FOUND;
13:
14:
15:    not_found = true;
16:    index = 0;
17:    while ( NULL != cmd_top[index].cmd_fnc )
18:    {
19:        if ( 0 == strcasecmp( cmd_top[index].name, argv[0] ) )
20:        {
21:            not_found = false;
22:            argv_new = &(argv[1]);
23:            argc_new = argc - 1;
24:            if ( 0 == argc_new ) argv_new = NULL;
25:            error_code = (* cmd_top[index].cmd_fnc )( board_id, argc_new, argv_new );
26:            if ( SUCCESS != error_code ) goto COMMAND_LINE_MAIN_TERMINATE;
27:            break;
28:        }
29:        index++;
30:    }
31:
32:    if ( not_found )
33:    {
34:        argv_new = &(argv[0]);
35:        argc_new = argc - 0;
36:        error_code = Command_Line_Register_Transaction( board_id, argc_new, argv_new );
37:        if ( SUCCESS != error_code ) goto COMMAND_LINE_MAIN_TERMINATE;
38:    }
39: COMMAND_LINE_MAIN_TERMINATE:
40:    return error_code;
41: }
```

3.2.1.122 Command_Line_Main__Board_Type_Status Function

C++

```
static enum CLM_BTS Command_Line_Main__Board_Type_Status(const char * cmd_string, int * board_id);
```

File

idi.c (see page 373)

Returns

0 = normal 1 = board type not required (i.e. 'board_id' parameter is a don't care) 2 = board type implied and determined here.

Description

brief Determine if this command requires specification of board type or not, or if a board type can be implied.

Body Source

```

1: static enum CLM_BTS Command_Line_Main__Board_Type_Status( const char * cmd_string, int *
board_id )
2: {
3:     int         index;
4:     enum CLM_BTS status;
5:     index      = 0;
6:     status     = CLM_BTS_NORMAL;
7:     *board_id = ID_BOARD_UNKNOWN;
8:     while ( NULL != cmd_top[index].cmd_fnc )
9:     {
16:
17:
18:     if      ( 0 == strcasecmp( "do", cmd_string ) )
19:     {
20:         *board_id = ID_IDO48; status = CLM_BTS_IMPLIED;
21:     }
22:     else if ( 0 == strcasecmp( "di", cmd_string ) )
23:     {
24:         *board_id = ID_IDI48; status = CLM_BTS_IMPLIED;
25:     }
26:
27:
28:
29:     if ( 0 == strcasecmp( cmd_top[index].name, cmd_string ) )
30:     {
31:         if ( false == cmd_top__board_type_required[index] )
32:         {
33:             if ( CLM_BTS_NORMAL == status ) status = CLM_BTS_NOT_REQUIRED;
34:         }
35:         return status;
36:     }
37:     index++;
38: }
39: return status;
40: }
```

3.2.1.123 Command_Line_Prefix Function

C++

```
int Command_Line_Prefix(int argc, char* argv[]);
```

File

idi.c (see page 373)

Returns

Index value to next argument in the list (i.e. updated index), or negative value indicating an error condition.

Description

brief

Body Source

```

1: int Command_Line_Prefix( int argc, char* argv[] )
2: {
3:     int    error_code;
4:     int    index;
5:     int    ai;
6:     int    argc_new;
7:     char ** argv_new;
8:
9:     if ( argc < 1 ) return 0;
10:
11:    index      = 0;
12:    ai         = 0;
13:    while ( NULL != cmd_prefix[index].cmd_fnc )
14:    {
15:        if ( 0 == strcmpi( cmd_prefix[index].name, argv[0] ) )
16:        {
17:            argv_new = &(argv[1]);
18:            argc_new = argc - 1;
19:            if ( 0 == argc_new ) argv_new = NULL;
20:            error_code = (* cmd_prefix[index].cmd_fnc )( argc_new, argv_new );
21:            if ( error_code < 0 ) goto COMMAND_LINE_PREFIX_TERMINATE;
22:            else ai = 1 + error_code;
23:            break;
24:        }
25:        index++;
26:    }
27:    return ai;
28: COMMAND_LINE_PREFIX_TERMINATE:
29:    return error_code;
30: }
```

3.2.1.124 Command_Line_Prefix_Irq Function

C++

```
static int Command_Line_Prefix_Irq(int argc, char* argv[]);
```

File

idi.c (see page 373)

Parameters

Parameters	Description
int argc	argument index in this case
char* argv[]	always null in this case

Description

brief

Body Source

```

1: static int Command_Line_Prefix_Irq( int    argc,
2:                                     char* argv[]
3:                                     )
```

```

4: {
5:   (void) argc;
6:   (void) argv;
7:
8:   global_irq.Please_install_handler_request = true;
9: #if(0)
10:  switch( board_id )
11:  {
12:    case ID_IDI48:
13:      ( ( struct idi_dataset * )  

board_dataset[board_id].dataset->irq.Please_install_handler_request = true;
14:      break;
15:    default:
16:      return -EC_BOARD_TYPE;
17:  }
18: #endif
19:
20: return 0;
21: }
```

3.2.1.125 Command_Line_Prefix_Loop Function

C++

```
static int Command_Line_Prefix_Loop(int argc, char* argv[]);
```

File

idi.c (see page 373)

Parameters

Parameters	Description
int argc	argument index in this case
char* argv[]	always null in this case

Description

brief

Body Source

```

1: static int Command_Line_Prefix_Loop( int argc,
2:                                     char* argv[ ]
3:                                     )
4: {
5:   int error_code;
6:   int index;
7:   int ai;
8:
9:   int argc_new;
10:  char ** argv_new;
11:
12:  global_loop_command = true;
13:
14:  index = 0;
15:  ai = 0;
16:  while ( NULL != cmd_loop[index].cmd_fnc )
17:  {
```

```

18: if ( 0 == strcasecmp( cmd_loop[index].name, argv[ai] ) )
19: {
20:
21:     argv_new = &(argv[ai + 1]);
22:     argc_new = argc - (ai + 1);
23:     if ( 0 == argc_new ) argv_new = NULL;
24:     error_code = (* cmd_loop[index].cmd_fnc )( argc_new, argv_new );
25:     if ( error_code < 0 ) goto COMMAND_LINE_PREFIX_LOOP_TERMINATE;
26:     else ai += 1 + error_code;
27:     index = 0;
28: }
29: else
30: {
31:     index++;
32: }
33: }
34: return ai;
35: COMMAND_LINE_PREFIX_LOOP_TERMINATE:
36: return error_code;
37: }
```

3.2.1.126 Command_Line_Prefix_Trace Function

C++

```
static int Command_Line_Prefix_Trace(int argc, char* argv[]);
```

File

idi.c (see page 373)

Parameters

Parameters	Description
int argc	argument index in this case
char* argv[]	always null in this case

Description

brief

Body Source

```

1: static int Command_Line_Prefix_Trace( int argc,
2:                                     char* argv[]
3:                                     )
4: {
5:     (void) argc;
6:     (void) argv;
7:     global_io_trace_enable = true;
8:     return SUCCESS;
9: }
```

3.2.1.127 Command_Line_Register_Transaction Function

C++

```
int Command_Line_Register_Transaction(int board_id, int argc, char* argv[]);
```

File

idi.c ([see page 373](#))

Returns

SUCCESS (0) if no errors encountered, otherwise errors are reported as a negative value.

Description

brief

Either reads or writes a register using the form: idi [

]

If

is not include, then it is assumed to be a read. If value is included, then a write to the specified register is made.

This function uses the definitions[] array which is global and built from IDI_REGISTER_SET_DEFINITION ([see page 350](#)) macro which is a nicely organized register list.

param[in] argc number of arguments including the executable file name param[in] argv list of string arguments lex'd from the command line

Body Source

```
1: int Command_Line_Register_Transaction( int board_id, int argc, char* argv[] )
2: {
3:     int         error_code;
4:     BOOL        valid_bank_reg;
5:     int         index;
6:     struct reg_definition * definition;
7:
8:
9:     definition = ( struct reg_definition * ) board_definition[board_id].definition;
10:
11:
12:    if ( argc < 1 ) return -EC_NOT_FOUND;
13:
14:    error_code = Register_Acronym_To_Row( board_id, argv[0], &index );
15:    if ( SUCCESS != error_code ) return error_code;
16:
17:
18:    valid_bank_reg = false;
19:    switch( board_id )
20:    {
21:        case ID_IDI48: if ( IDI_BANK == definition[index].symbol ) valid_bank_reg = true;
22:        break;
23:        case ID_IDO48: if ( IDO_BANK == definition[index].symbol ) valid_bank_reg = true;
24:        break;
25:    }
26:
```

```
28: if ( valid_bank_reg )
29: {
30:     if ( argc < 2 )
31:     {
32:         uint8_t value;
33:         error_code = IO_Read_U8( board_id, __LINE__, definition[index].symbol, &value );
34:         if ( SUCCESS == error_code )
35:         {
36:             printf( "RD: %s=%s (0x%02X)\n", definition[index].acronym, Bank_Symbol_To_Name(
board_id, ((int) value) ), value );
37:         }
38:     }
39:     else
40:     {
41:         uint8_t value;
42:         int bank;
43:
44:         error_code = Bank_Name_To_Symbol( board_id, argv[1], &bank );
45:         if ( SUCCESS != error_code )
46:         {
47:             value = (uint8_t) strtol( argv[1], NULL, 0 );
48:         }
49:         else
50:         {
51:             value = (uint8_t) bank;
52:         }
53:         error_code = IO_Write_U8( board_id, __LINE__, definition[index].symbol, value );
54:         if ( SUCCESS == error_code )
55:         {
56:             printf( "WR: %s=0x%02X\n", definition[index].acronym, value );
57:         }
58:     }
59: }
60: else
61: {
62:     if ( argc < 2 )
63:     {
64:         uint8_t value;
65:         error_code = IO_Read_U8( board_id, __LINE__, definition[index].symbol, &value );
66:         if ( SUCCESS == error_code )
67:         {
68:             printf( "RD: %s=0x%02X\n", definition[index].acronym, value );
69:         }
70:     }
71:     else
72:     {
73:         uint8_t value;
74:         value = (uint8_t) strtol( argv[1], NULL, 0 );
75:         error_code = IO_Write_U8( board_id, __LINE__, definition[index].symbol, value );
76:         if ( SUCCESS == error_code )
77:         {
78:             printf( "WR: %s=0x%02X\n", definition[index].acronym, value );
79:         }
80:     }
81: }
82:
83:     return SUCCESS;
84: }
```

3.2.1.128 Command_Line_Set Function

C++

```
int Command_Line_Set(int board_id, int argc, char* argv[ ]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```
1: int Command_Line_Set( int board_id, int argc, char* argv[ ] )
2: {
3:     int     error_code;
4:     int     index;
5:     int     argc_new;
6:     char ** argv_new;
7:     struct board_dataset * dataset;
8:     (void) board_id;
9:
10:    dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
11:    error_code = -EC_SYNTAX;
12:
13:    if ( argc < 1 )
14:    {
15:        index = 0;
16:        while ( NULL != cmd_set[index].cmd_fnc )
17:        {
18:            argv_new = &(argv[1]);
19:            argc_new = argc - 1;
20:            if ( 0 == argc_new ) argv_new = NULL;
21:            dataset->set__suppress_io_activity = true;
22:            error_code = (* cmd_set[index].cmd_fnc )( board_id, argc_new, argv_new );
23:            dataset->set__suppress_io_activity = false;
24:            index++;
25:        }
26:    }
27:    else
28:    {
29:        index = 0;
30:        while ( NULL != cmd_set[index].cmd_fnc )
31:        {
32:            if ( 0 == strcasecmp( cmd_set[index].name, argv[0] ) )
33:            {
34:                argv_new = &(argv[1]);
35:                argc_new = argc - 1;
36:                if ( 0 == argc_new ) argv_new = NULL;
37:                dataset->set__suppress_io_activity = true;
38:                error_code = (* cmd_set[index].cmd_fnc )( board_id, argc_new, argv_new );
39:                dataset->set__suppress_io_activity = false;
```

```

40:     break;
41:   }
42:   index++;
43: }
44: }
45: return error_code;
46: }
```

3.2.1.129 Command_Line_SPI Function

C++

```
int Command_Line_SPI(int board_id, int argc, char* argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: int Command_Line_SPI( int board_id, int argc, char* argv[ ] )
2: {
3:   int   error_code;
4:   int   index;
5:   int    argc_new;
6:   char ** argv_new;
7:
8:   error_code = -EC_SYNTAX;
9:
10:  if ( argc < 1 ) return -EC_NOT_FOUND;
11:
12:  index = 0;
13:  while ( NULL != cmd_spi[index].cmd_fnc )
14:  {
15:    if ( 0 == strcmpi( cmd_spi[index].name, argv[0] ) )
16:    {
17:      argv_new = &(argv[1]);
18:      argc_new = argc - 1;
19:      if ( 0 == argc_new ) argv_new = NULL;
20:
21:    {
22:      struct board_dataset * dataset;
23:      dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
24:      if ( IO_TRACE_START_SPI == dataset->trace.start )
25:      {
26:        dataset->trace.active = true;
27:      }
28:    }
29:
30:    error_code = (* cmd_spi[index].cmd_fnc )( board_id, argc_new, argv_new );
31:    break;
32:  }
```

```
33:     index++;
34: }
35: return error_code;
36: }
```

3.2.1.130 Command_Line_Wait Function

C++

```
int Command_Line_Wait(int board_id, int argc, char* argv[]);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```
1: int Command_Line_Wait( int board_id, int argc, char* argv[ ] )
2: {
3:     (void) board_id;
4:     (void) argc;
5:     (void) argv;
6:
7:     if ( global_loop_command ) return SUCCESS;
8:     global_loop_command = true;
9:     return SUCCESS;
10: }
```

3.2.1.131 EC_Code_To_Human_Readable Function

C++

```
const char * EC_Code_To_Human_Readable(EC_ENUM error_code);
```

File

idi.c (see page 373)

Description

This is function EC_Code_To_Human_Readable.

Body Source

```
1: const char * EC_Code_To_Human_Readable( EC_ENUM error_code )
2: {
3:     int index;
4:
5:     if ( error_code < 0 ) error_code = -error_code;
```

```

6:
7:     index = 0;
8:     while( NULL != ec_human_readable[index].message )
9:     {
10:         if ( ((EC_ENUM) error_code) == ec_human_readable[index].error_code )
11:         {
12:             return ec_human_readable[index].message;
13:         }
14:         index++;
15:     }
16:     return ec_unknown;
17: }
```

3.2.1.132 FPGA_Date_Time_Information Function

C++

`int FPGA_Date_Time_Information(int board_id, uint32_t * date_svn, uint32_t * date_compilation);`

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: int FPGA_Date_Time_Information( int board_id, uint32_t * date_svn, uint32_t *
date_compilation )
2: {
3:     int     location_data;
4:     int     location_index;
5:     int     state;
6:     size_t   index;
7:     size_t   count;
8:     BOOL    fpga_data_available;
9:     const uint8_t index_test_list[] = { 0x05, 0x0A, 0x00 };
10:    uint8_t   scratch[16] = { 0x00 };
11:    uint8_t   block_offset;
12:    uint8_t   block_type;
13:    uint32_t  scratch_u32;
14:    #define FPGA_DATE_SIZE 4
15:    struct board_dataset * dataset;
16:    dataset = (struct board_dataset *) board_definition[board_id].dataset;
17:
18:    if ( MODE_JUMPERS_0 == dataset->mode_jumpers )
19:    {
20:        return -EC_MODE_LEGACY;
21:    }
22:
23:
24:    location_data     = board_definition[board_id].fpga_data_symbol;
25:    location_index    = board_definition[board_id].fpga_index_symbol;
26:    fpga_data_available = true;
```

```
27:
28:     count      = 0;
29:     while ( 0x00 != index_test_list[count] )
30:     {
31:         IO_Write_U8( board_id, __LINE__, location_index, index_test_list[count] );
32:         IO_Read_U8( board_id, __LINE__, location_index, &(scratch[count]) );
33:         count++;
34:     }
35:     for ( index = 0; index < count; index++ )
36:     {
37:         if ( index_test_list[index] != scratch[index] )
38:         {
39:             fpga_data_available = false;
40:             break;
41:         }
42:     }
43:
44:     if ( false == fpga_data_available )
45:     {
46:         *date_svn = 0;
47:         *date_compilation = 0;
48:         return SUCCESS;
49:     }
50:
51:     state      = 0;
52:     index      = 0;
53:     block_offset = 0;
54:     while( 3 != state )
55:     {
56:         switch ( state )
57:         {
58:             case 0:
59:                 IO_Write_U8( board_id, __LINE__, location_index, (uint8_t) index );
60:                 index++;
61:                 IO_Read_U8( board_id, __LINE__, location_data, &block_offset );
62:
63:                 IO_Write_U8( board_id, __LINE__, location_index, (uint8_t) index );
64:                 index++;
65:                 IO_Read_U8( board_id, __LINE__, location_data, &block_type );
66:
67:                 switch( block_type )
68:                 {
69:                     case 's':
70:                     case 'S':
71:                         state = 1;  count = 0;
72:                         break;
73:                     case 'c':
74:                     case 'C':
75:                         state = 2;  count = 0;
76:                         break;
77:                     case 0x00:
78:                     case 0xFF:
79:                         state = 3;
80:                         count = 0;
81:                         break;
82:                     default:
83:                         break;
84:                 }
85:                 break;
86:             case 1:
87:                 IO_Write_U8( board_id, __LINE__, location_index, (uint8_t) index );
88:                 index++; count++;
89:                 IO_Read_U8( board_id, __LINE__, location_data, &(scratch[FPGA_DATE_SIZE - count]) );
90:             if ( FPGA_DATE_SIZE == count )
```

```

91:      {
92:        size_t i;
93:        scratch_u32 = 0;
94:        for ( i = 0; i < FPGA_DATE_SIZE; i++ )
95:        {
96:          scratch_u32 = ( scratch_u32 << 8 ) + ( (uint32_t) scratch[i] );
97:        }
98:        *date_svn = scratch_u32;
99:        state = 0;
100:      }
101:      break;
102: case 2:
103:   IO_Write_U8( board_id, __LINE__, location_index, (uint8_t) index );
104:   index++; count++;
105:   IO_Read_U8( board_id, __LINE__, location_data, &(scratch[FPGA_DATE_SIZE - count]) );
106:   if ( FPGA_DATE_SIZE == count )
107:   {
108:     size_t i;
109:     scratch_u32 = 0;
110:     for ( i = 0; i < FPGA_DATE_SIZE; i++ )
111:     {
112:       scratch_u32 = ( scratch_u32 << 8 ) + ( (uint32_t) scratch[i] );
113:     }
114:     *date_compilation = scratch_u32;
115:     state = 0;
116:   }
117:   break;
118: }
119: }
120: return SUCCESS;
121: #undef FPGA_DATE_SIZE
122: }
```

3.2.1.133 FRAM__Chip_Select_Route_Override Function

C++

```
static int FRAM__Chip_Select_Route_Override(int board_id);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief FRAM SPI chip select over-ride mechanism. Temporarily used to set the chip select channel to what we need based on "set frams".

Body Source

```

1: static int FRAM__Chip_Select_Route_Override( int board_id )
2: {
3:   struct board_dataset * dataset;
4:   dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
5:
6:   #define FUNCTION_NAME__ "FRAM__Chip_Select_Route_Override"
```

```

7: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
8: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );
9: #endif
10: #undef FUNCTION_NAME__
11:
12: SPI_Configuration_Chip_Select_Route_Get( board_id, &(dataset->fram_spi_cs_route_backup) );
13:
14: SPI_Configuration_Chip_Select_Route_Set( board_id, dataset->fram_cs_default );
15: return SUCCESS;
16: }
```

3.2.1.134 FRAM__Chip_Select_Route_Restore Function

C++

```
static int FRAM__Chip_Select_Route_Restore(int board_id);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief FRAM SPI chip select restore mechanism. Returns the chip select to original SPI configuration settings.

Body Source

```

1: static int FRAM__Chip_Select_Route_Restore( int board_id )
2: {
3:     struct board_dataset * dataset;
4:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
5:
6: #define FUNCTION_NAME__ "FRAM__Chip_Select_Route_Restore"
7: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
8: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );
9: #endif
10: #undef FUNCTION_NAME__
11:
12: SPI_Configuration_Chip_Select_Route_Set( board_id, dataset->fram_spi_cs_route_backup );
13: return SUCCESS;
14: }
```

3.2.1.135 FRAM__Memory_Read Function

C++

```
int FRAM__Memory_Read(int board_id, uint16_t address, size_t count, uint8_t * buffer);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Reads data from FRAM memory to the output buffer (see page 325).

param[in] address Starting FRAM address param[in] count Number of bytes to transfer param[out] buffer (see page 325)
Destination buffer (see page 325) in which to store the data

Body Source

```

1: int FRAM__Memory_Read( int board_id, uint16_t address, size_t count, uint8_t * buffer )
2: {
3:     int     error_code;
4:
5:     uint8_t    tx_buf[3] = { 0x03, 0x00, 0x00 };
6:
7:
8: #define FUNCTION_NAME__ "FRAM__Memory_Read"
9: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
10:    IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );
11: #endif
12: #undef FUNCTION_NAME__
13:
14:
15: #if defined( SPI_PRINT_DEBUG )
16:    printf( "*** FRAM__Memory_Read: Entry.\n" );
17: #endif
18:
19:    tx_buf[1] = (uint8_t)( address >> 8 );
20:    tx_buf[2] = (uint8_t)( address & 0xFF );
21:    FRAM__Chip_Select_Route_Override( board_id );
22:
23:
24:
25:
26:    error_code = SPI_Configuration_Chip_Select_Behavior_Set( board_id, CSB_SOFTWARE );
27:    if ( SUCCESS != error_code ) goto FRAM__Memory_Read_Error_Exit;
28:
29:    SPI_Commit( board_id, 0xFF );
30:
31: #if defined( SPI_PRINT_DEBUG )
32:    printf( "*** FRAM__Memory_Read: Send FRAM Write Command.\n" );
33: #endif
34:
35:    error_code = SPI_Data_Write_Read( board_id, 3, sizeof( uint8_t ), tx_buf, 0, NULL );
36:    if ( SUCCESS != error_code ) goto FRAM__Memory_Read_Error_Exit;
37:
38: #if defined( SPI_PRINT_DEBUG )
39:    printf( "*** FRAM__Memory_Read: count=%u\n", count );
40: #endif
41:
42:    error_code = SPI_Data_Write_Read( board_id, sizeof( uint8_t ), 0, NULL, count, buffer );
43:    if ( SUCCESS != error_code ) goto FRAM__Memory_Read_Error_Exit;
44:
45:
46:    SPI_Commit( board_id, 0x00 );
47:
48:
49:
```

```

50:
51: FRAM__Chip_Select_Route_Restore( board_id );
52: return SUCCESS;
53:
54: FRAM__Memory_Read_Error_Exit:
55: SPI_Commit( board_id, 0x00 );
56:
57: #define FUNCTION_NAME__ "FRAM__Memory_Read"
58: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
59: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, error_code );
60: #endif
61: #undef FUNCTION_NAME__
62:
63: FRAM__Chip_Select_Route_Restore( board_id );
64: return error_code;
65: }
```

3.2.1.136 FRAM__Memory_Write Function

C++

```
int FRAM__Memory_Write(int board_id, uint16_t address, size_t count, uint8_t * buffer);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Writes data from buffer (see page 325) to FRAM memory.

param[in] address Starting FRAM address param[in] count Number of bytes to transfer param[out] buffer (see page 325)
Source buffer (see page 325) from which data will be transferred to FRAM

Body Source

```

1: int FRAM__Memory_Write( int board_id, uint16_t address, size_t count, uint8_t * buffer )
2: {
3:     int     error_code;
4:
5:     uint8_t    tx_buf[3] = { 0x02, 0x00, 0x00 };
6:
7:     #define FUNCTION_NAME__ "FRAM__Memory_Write"
8:     #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
9:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );
10:    #endif
11:    #undef FUNCTION_NAME__
12:
13:    tx_buf[1] = (uint8_t)( address >> 8 );
14:    tx_buf[2] = (uint8_t)( address & 0xFF );
15:
16:    FRAM__Write_Enable_Latch_Set( board_id );
17:    FRAM__Chip_Select_Route_Override( board_id );
18:
19:
20:
```

```

21:
22: error_code = SPI_Configuration_Chip_Select_Behavior_Set( board_id, CSB_SOFTWARE );
23: if ( SUCCESS != error_code ) goto FRAM__Memory_Write_Error_Exit;
24:
25: SPI_Commit( board_id, 0xFF );
26:
27: error_code = SPI_Data_Write_Read( board_id, 3, sizeof( uint8_t ), tx_buf, 0, NULL );
28: if ( SUCCESS != error_code ) goto FRAM__Memory_Write_Error_Exit;
29:
30: error_code = SPI_Data_Write_Read( board_id, sizeof( uint8_t ), count, buffer, 0, NULL );
31: if ( SUCCESS != error_code ) goto FRAM__Memory_Write_Error_Exit;
32:
33:
34: SPI_Commit( board_id, 0x00 );
35:
36:
37:
38:
39: FRAM__Chip_Select_Route_Restore( board_id );
40: return SUCCESS;
41:
42: FRAM__Memory_Write_Error_Exit:
43: SPI_Commit( board_id, 0x00 );
44:
45: #define FUNCTION_NAME__ "FRAM__Memory_Write"
46: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
47: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, error_code );
48: #endif
49: #undef FUNCTION_NAME__
50:
51: FRAM__Chip_Select_Route_Restore( board_id );
52: return error_code;
53: }
```

3.2.1.137 FRAM__Read_ID Function

C++

```
int FRAM__Read_ID(int board_id, uint32_t * id);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

param[out] id The 32-bit ID register read from the FRAM. This appears to be only available with Fujitsu parts.

Body Source

```

1: int FRAM__Read_ID( int board_id, uint32_t * id )
2: {
3:     int error_code;
4:     uint8_t tx_buf[5] = { 0x9F, 0x00, 0x00, 0x00, 0x00 };
```

```

5:  uint8_t rx_buf[5];
6:
7: #define FUNCTION_NAME__ "FRAM__Read_ID"
8: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
9: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );
10: #endif
11: #undef FUNCTION_NAME__
12:
13: FRAM__Chip_Select_Route_Override( board_id );
14: SPI_Configuration_Chip_Select_Behavior_Set( board_id, CSB_BUFFER );
15: error_code = SPI_Data_Write_Read( board_id, 5, 1, tx_buf, 1, rx_buf );
16: if ( SUCCESS != error_code ) goto FRAM__Read_ID_Error_Exit;
17:
18: {
19:     uint32_t id_scratch = 0;
20:     size_t index;
21:
22:     for ( index = 1; index < 5; index++ )
23:     {
24:         id_scratch = ( id_scratch << 8 ) | ( (uint32_t) rx_buf[index] );
25:     }
26:     *id = id_scratch;
27: }
28: #if defined( IDI_FRAM_PRINT_DEBUG )
29: {
30:     int index;
31:     printf( "FRAM__Read_ID:" );
32:     for ( index = 0; index < 5; index++ )
33:     {
34:         printf( " 0x%02X", rx_buf[index] );
35:     }
36:     printf( "\n" );
37: }
38: #endif
39: FRAM__Chip_Select_Route_Restore( board_id );
40: return SUCCESS;
41: FRAM__Read_ID_Error_Exit:
42:
43: #define FUNCTION_NAME__ "FRAM__Read_ID"
44: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
45: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, error_code );
46: #endif
47: #undef FUNCTION_NAME__
48:
49: FRAM__Chip_Select_Route_Restore( board_id );
50: return error_code;
51: }
```

3.2.1.138 FRAM__Read_Status_Register Function

C++

```
int FRAM__Read_Status_Register(int board_id, uint8_t * status);
```

File

idi.c (see page 373)

Parameters

Parameters	Description
uint8_t * status	A pointer to the destination location of the status data.

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Read the FRAM status register and output the value.

Body Source

```

1: int FRAM__Read_Status_Register( int board_id, uint8_t * status )
2: {
3:     int error_code;
4:     uint8_t tx_buf[2] = { 0x05, 0x00 };
5:     uint8_t rx_buf[2];
6:
7: #define FUNCTION_NAME__ "FRAM__Read_Status_Register"
8: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
9:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );
10: #endif
11: #undef FUNCTION_NAME__
12:
13:     FRAM__Chip_Select_Route_Override( board_id );
14:
15:     SPI_Configuration_Chip_Select_Behavior_Set( board_id, CSB_BUFFER );
16:     error_code = SPI_Data_Write_Read( board_id, 2, 1, tx_buf, 1, rx_buf );
17:     FRAM__Chip_Select_Route_Restore( board_id );
18:     if ( SUCCESS != error_code ) return error_code;
19:     *status = rx_buf[1];
20:
21: #if defined( IDI_FRAM_PRINT_DEBUG )
22: {
23:     int index;
24:     printf( "FRAM__Read_Status_Register:" );
25:     for ( index = 0; index < 2; index++ )
26:     {
27:         printf( " 0x%02X", rx_buf[index] );
28:     }
29:     printf( "\n" );
30: }
31: #endif
32: return SUCCESS;
33: }
```

3.2.1.139 FRAM__Write_Disable Function

C++

```
int FRAM__Write_Disable(int board_id);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief FRAM Write Latch disable (or clear) command (WRDI).

Body Source

```

1: int FRAM__Write_Disable( int board_id )
2: {
3:     int error_code;
4:     uint8_t tx_buf[1] = { 0x04 };
5:
6: #define FUNCTION_NAME__ "FRAM__Write_Disable"
7: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
8:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );
9: #endif
10: #undef FUNCTION_NAME__
11:
12: FRAM__Chip_Select_Route_Override( board_id );
13:
14: SPI_Configuration_Chip_Select_Behavior_Set( board_id, CSB_BUFFER );
15: error_code = SPI_Data_Write_Read( board_id, 1, 1, tx_buf, 0, NULL );
16: FRAM__Chip_Select_Route_Restore( board_id );
17: return error_code;
18: }
```

3.2.1.140 FRAM__Write_Enable_Latch_Set Function

C++

```
int FRAM__Write_Enable_Latch_Set(int board_id);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief FRAM Write Enable Latch Set command (WREN)

Body Source

```

1: int FRAM__Write_Enable_Latch_Set( int board_id )
2: {
3:     int error_code;
4:     uint8_t tx_buf[1] = { 0x06 };
5:
6: #define FUNCTION_NAME__ "FRAM__Write_Enable_Latch_Set"
7: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
8:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );
9: #endif
```

```

10: #undef FUNCTION_NAME__
11:
12: FRAM__Chip_Select_Route_Override( board_id );
13:
14: SPI_Configuration_Chip_Select_Behavior_Set( board_id, CSB_BUFFER );
15: error_code = SPI_Data_Write_Read( board_id, 1, 1, tx_buf, 0, NULL );
16: FRAM__Chip_Select_Route_Restore( board_id );
17: return error_code;
18: }
```

3.2.1.141 FRAM__Write_Status_Register Function

C++

```
int FRAM__Write_Status_Register(int board_id, uint8_t status);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Write to the FRAM status register.

param[in] FRAM status value to be written.

Body Source

```

1: int FRAM__Write_Status_Register( int board_id, uint8_t status )
2: {
3:     int error_code;
4:     uint8_t tx_buf[2] = { 0x01, 0x00 };
5:     uint8_t rx_buf[2];
6:
7: #define FUNCTION_NAME__ "FRAM__Write_Status_Register"
8: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
9:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );
10: #endif
11: #undef FUNCTION_NAME__
12:
13: FRAM__Write_Enable_Latch_Set( board_id );
14:
15: tx_buf[1] = status;
16: FRAM__Chip_Select_Route_Override( board_id );
17:
18: SPI_Configuration_Chip_Select_Behavior_Set( board_id, CSB_BUFFER );
19: error_code = SPI_Data_Write_Read( board_id, 2, 1, tx_buf, 1, rx_buf );
20: FRAM__Chip_Select_Route_Restore( board_id );
21: if ( SUCCESS != error_code ) return error_code;
22:
23: return SUCCESS;
24: }
```

3.2.1.142 FRAM_File_To_Memory Function

C++

```
int FRAM_File_To_Memory(int board_id, uint16_t address, size_t length, FILE * binary);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```
1: int FRAM_File_To_Memory( int board_id, uint16_t address, size_t length, FILE * binary )
2: {
3:     int error_code;
4:     size_t count_read;
5:     size_t count_total;
6:     size_t count_actual;
7:     struct board_dataset * dataset      = ( struct board_dataset * )
board_definition[board_id].dataset;
8:
9: #define FUNCTION_NAME__ "FRAM_File_To_Memory"
10: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
11:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );
12: #endif
13: #undef FUNCTION_NAME__
14:
15: count_total = 0;
16: count_read  = FRAM_BLOCK_SIZE;
17: if ( 0 == length )
18: {
19:     do
20:     {
21:         count_actual = fread( dataset->spi.fram_block, 1, count_read, binary );
22:         error_code   = FRAM__Memory_Write( board_id, address, count_read,
dataset->spi.fram_block );
23:         if ( SUCCESS != error_code ) goto FRAM_File_To_Memory_Error_Exit;
24:         count_total += count_actual;
25:         if ( count_actual != count_read ) count_read = 0;
26:     } while ( count_read > 0 );
27: }
28: else
29: {
30:     if ( length < count_read ) count_read = length;
31:     do
32:     {
33:         count_actual = fread( dataset->spi.fram_block, 1, count_read, binary );
34:         error_code   = FRAM__Memory_Write( board_id, address, count_read,
dataset->spi.fram_block );
35:         if ( SUCCESS != error_code ) goto FRAM_File_To_Memory_Error_Exit;
36:         count_total += count_actual;
```

```

37:     length      -= count_actual;
38:     if ( count_actual != count_read ) count_read = 0;
39:     if ( length < count_read ) count_read = length;
40: } while ( count_read > 0 );
41: }
42: return SUCCESS;
43:
44: FRAM_File_To_Memory_Error_Exit:
45:
46: #define FUNCTION_NAME__ "FRAM_File_To_Memory"
47: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
48: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, error_code );
49: #endif
50: #undef FUNCTION_NAME__
51:
52: return error_code;
53: }
```

3.2.1.143 FRAM_Memory_To_File Function

C++

```
int FRAM_Memory_To_File(int board_id, uint16_t address, size_t length, FILE * binary);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: int FRAM_Memory_To_File( int board_id, uint16_t address, size_t length, FILE * binary )
2: {
3:     int error_code;
4:     size_t block_count;
5:     size_t block_remainder;
6:     struct board_dataset * dataset     = ( struct board_dataset * )
board_definition[board_id].dataset;
7:
8: #define FUNCTION_NAME__ "FRAM_Memory_To_File"
9: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
10: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );
11: #endif
12: #undef FUNCTION_NAME__
13:
14: block_count = length / ((size_t) FRAM_BLOCK_SIZE);
15: block_remainder = length - ( block_count * ((size_t) FRAM_BLOCK_SIZE) );
16:
17: while ( block_count > 0 )
18: {
19:     error_code = FRAM__Memory_Read( board_id, address, ((size_t) FRAM_BLOCK_SIZE),
dataset->spi.fram_block );
20:     if ( SUCCESS != error_code ) goto FRAM_Memory_To_File_Error_Exit;
```

```

21:   fwrite( dataset->spi.fram_block, 1, ((size_t) FRAM_BLOCK_SIZE), binary );
22:   address += FRAM_BLOCK_SIZE;
23:   block_count--;
24: }
25:
26: if ( block_remainder > 0 )
27: {
28:   error_code = FRAM__Memory_Read( board_id, address, block_remainder,
dataset->spi.fram_block );
29:   if ( SUCCESS != error_code ) goto FRAM_Memory_To_File_Error_Exit;
30:   fwrite( dataset->spi.fram_block, 1, block_remainder, binary );
31: }
32: return SUCCESS;
33:
34: FRAM_Memory_To_File_Error_Exit:
35:
36: #define FUNCTION_NAME__ "FRAM_Memory_To_File"
37: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
38: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, error_code );
39: #endif
40: #undef FUNCTION_NAME__
41:
42: return error_code;
43: }
```

3.2.1.144 FRAM_Report Function

C++

```
int FRAM_Report(int board_id, uint16_t address, size_t length, FILE * out);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: int FRAM_Report( int board_id, uint16_t address, size_t length, FILE * out )
2: {
3:   int error_code;
4:   const int block_size = HEX_DUMP_BYTES_PER_LINE;
5:   size_t block_count;
6:   size_t block_remainder;
7:   struct board_dataset * dataset     = ( struct board_dataset * )
board_definition[board_id].dataset;
8:
9: #define FUNCTION_NAME__ "FRAM_Report"
10: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
11: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );
12: #endif
13: #undef FUNCTION_NAME__
```

```

15: #if defined( SPI_PRINT_XFR_DEBUG )
16:   fprintf( out, "FRAM using SPI CS channel %d\n", dataset->fram_cs_default ? 1 : 0 );
17: #endif
18:
19: block_count = ((size_t) length) / block_size;
20: block_remainder = ((size_t) length) - ( block_count * block_size );
21:
22: while ( block_count > 0 )
23: {
24:   error_code = FRAM_Memory_Read( board_id, address, block_size, dataset->spi.fram_block );
25:   if ( SUCCESS != error_code ) goto FRAM_Report_Error_Exit;
26:   error_code = Hex_Dump_Line( address, block_size, dataset->spi.fram_block, out );
27:   if ( SUCCESS != error_code ) goto FRAM_Report_Error_Exit;
28:   address += block_size;
29:   block_count--;
30: }
31:
32: if ( block_remainder > 0 )
33: {
34:   error_code = FRAM_Memory_Read( board_id, address, block_remainder,
dataset->spi.fram_block );
35:   if ( SUCCESS != error_code ) goto FRAM_Report_Error_Exit;
36:   error_code = Hex_Dump_Line( address, block_remainder, dataset->spi.fram_block, out );
37:   if ( SUCCESS != error_code ) goto FRAM_Report_Error_Exit;
38: }
39:
40: return SUCCESS;
41:
42: FRAM_Report_Error_Exit:
43:
44: #define FUNCTION_NAME__ "FRAM_Report"
45: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
46:   IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, error_code );
47: #endif
48: #undef FUNCTION_NAME__
49:
50: return error_code;
51: }
```

3.2.1.145 FRAM_Set Function

This function will be used when creating a memory pool so that as blocks are allocated one can determine if we have an issue outside of any allocated space (i.e. overflows and so on).

param[in] cfg pass in the configuration to be written to hardware. return a nonzero if successful, else return zero.

**

- @brief
- Writes a repeating pattern to the entire FRAM memory array. the pattern is obtained from the buffer (see page 325). If the buffer (see page 325) is NULL, then all zeros are written to the FRAM.
- *
- @param[in] count Length in bytes of the pattern found within the buffer (see page 325)
- @param[in] buffer (see page 325) input buffer (see page 325) containing the repeat pattern to be written to FRAM

- @return A zero (SUCCESS) is returned if successful, otherwise a negative error
- code is returned.

C++

```
int FRAM_Set(int board_id, size_t count, uint8_t * buffer);
```

File

idi.c (see page 373)

Body Source

```
1: int FRAM_Set( int board_id, size_t count, uint8_t * buffer )
2: {
3:     int     error_code;
4:     size_t   block_count;
5:     size_t   block_remainder;
6:     uint16_t address;
7: #if defined( IDI_FRAM_PRINT_DEBUG )
8:     size_t   index;
9: #endif
10:
11:    struct board_dataset * dataset      = ( struct board_dataset * )
board_definition[board_id].dataset;
12:
13: #define FUNCTION_NAME__ "FRAM_Set"
14: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
15:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );
16: #endif
17: #undef FUNCTION_NAME__
18:
19:
20:
21:
22:    address = 0;
23:    error_code = SUCCESS;
24:
25:    if ( count > 1 )
26:    {
27:        block_count      = ((size_t) FRAM_DENSITY_BYTES) / count;
28:        block_remainder = ((size_t) FRAM_DENSITY_BYTES) - ( block_count * count );
29:    }
30:    else
31:    {
32:        int     index;
33:        const int block_size = FRAM_BLOCK_SIZE;
34:
35:        if ( NULL == buffer )
36:        {
37:            for ( index = 0; index < block_size; index++ ) dataset->spi.fram_block[index]= 0;
38:        }
39:        else
40:        {
41:            for ( index = 0; index < block_size; index++ ) dataset->spi.fram_block[index]=
buffer[0];
42:        }
43:        block_count      = ((size_t) FRAM_DENSITY_BYTES) / block_size;
44:        block_remainder = ((size_t) FRAM_DENSITY_BYTES) - ( block_count * block_size );
45:        buffer           = dataset->spi.fram_block;
46:        count            = block_size;
47:    }
48:
```

```

49: #if defined( IDI_FRAM_PRINT_DEBUG )
50: index = 0;
51: # if defined( __MSDOS__ )
52:     printf( "FRAM_Set: count=%d, block_count=%d, block_remainder=%d\n", count, block_count,
53: block_remainder );
53: # else
54:     printf( "FRAM_Set: count=%lu, block_count=%lu, block_remainder=%lu\n", count,
55: block_count, block_remainder );
55: #endif
56: #endif
57:
58: while ( block_count > 0 )
59: {
60:     error_code = FRAM_Memory_Write( board_id, address, count, buffer );
61:     if ( SUCCESS != error_code ) goto FRAM_Set_Error_Exit;
62:     address += (uint16_t) count;
63:     block_count--;
64:
65: #if defined( IDI_FRAM_PRINT_DEBUG )
66:     if ( index < 16 )
67:     {
68: # if defined( __MSDOS__ )
69:         printf( "FRAM_Set: block_count=%d, address=%d\n", block_count, address );
70: # else
71:         printf( "FRAM_Set: block_count=%lu, block_remainder=%d\n", block_count, address );
72: #endif
73:         index++;
74:     }
75: #endif
76:
77: }
78:
79: if ( block_remainder > 0 )
80: {
81:     error_code = FRAM_Memory_Write( board_id, address, block_remainder, buffer );
82:     if ( SUCCESS != error_code ) goto FRAM_Set_Error_Exit;
83: }
84:
85:
86: return SUCCESS;
87:
88: FRAM_Set_Error_Exit:
89:
90: #define FUNCTION_NAME__ "FRAM_Set"
91: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
92:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, error_code );
93: #endif
94: #undef FUNCTION_NAME__
95:
96: return error_code;
97: }
```

3.2.1.146 Help Function

C++

```
static void Help(BOOL skip_pause, FILE * out);
```

File

idi.c (see page 373)

Description

brief Outputs a help listing to the user.

Body Source

```

1: static void Help( BOOL skip_pause, FILE * out )
2: {
3:   size_t line;
4:   int top;
5:
6:   line = 0;
7:   fprintf( out, "\n" );
8:   fprintf( out, "STDBus General Test Code\n" );
9:   fprintf( out, "Apex Embedded Systems\n" );
10:  fprintf( out, "SVN software revision: " SVN_REV "\n" );
11:  fprintf( out, "Supporting:\n" );
12:  line += 5;
13:
14:  IDI_Help( &line, out );
15:  IDO_Help( &line, out );
16:
17:  fprintf( out, "\n" );
18:  fprintf( out, "help - outputs help information\n" );
19:  line += 2;
20:
21:  top = 0;
22:  while ( NULL != cmd_prefix[top].name )
23:  {
24:    fprintf( out, "\n" );
25:    fprintf( out, "%-4s - %s\n", cmd_prefix[top].name, cmd_prefix[top].help );
26:    top++;
27:    line += 2;
28:    Help_Pause_Helper( skip_pause, &line );
29:  }
30:
31:  Help_Output( skip_pause, 0, &line, (struct command_line_board *) cmd_top , out );
32:  fprintf( out, "\n" );
33: }
```

3.2.1.147 Help_Output Function

C++

```
static void Help_Output(BOOL skip_pause, int depth, size_t * line, struct command_line_board * table, FILE * out);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static void Help_Output( BOOL      skip_pause,
2:                         int       depth,
```

```

3:      size_t *      line,
4:      struct command_line_board *  table,
5:      FILE *        out
6:      )
7:  {
8:  size_t   row;
9:  int     index;
10: const char * prefix = { "           " };
11:
12: row = 0;
13: while ( NULL != table[row].name )
14: {
15:   if ( 0 == depth )
16:   {
17:     fprintf( out, "%-4s", table[row].name );
18:   }
19:   else
20:   {
21:     for ( index = 0; index < depth; index++ ) fprintf( out, "%s", prefix );
22:     fprintf( out, "%-8s", table[row].name );
23:   }
24:   fprintf( out, " - %s\n", table[row].help );
25:   *line += 1;
26: #if defined( __MSDOS__ )
27:   Help_Pause_Helper( skip_pause, line );
28: #endif
29:   if ( NULL != table[row].link )
30:   {
31:     Help_Output( skip_pause, depth + 1, line, table[row].link, out );
32:     fprintf( out, "\n" );
33:     *line += 1;
34:   }
35:   row++;
36: }
37: }
```

3.2.1.148 Help_Pause_Helper Function

C++

```
static void Help_Pause_Helper(BOOL skip_pause, size_t * line);
```

File

idi.c (see page 373)

Description

brief Outputs a help listing to the user.

Body Source

```

1: static void Help_Pause_Helper( BOOL skip_pause, size_t * line )
2: {
3:   if ( true == skip_pause ) return;
4:   if ( *line > 20 )
5:   {
6:     printf( "Press any key for more....." );
7:     while ( !Character_Get( NULL ) ) { }
8:     printf( "\r" );
9:     *line = 0;
```

```
10: }
11: }
```

3.2.1.149 Hex_Dump_Line Function

C++

```
int Hex_Dump_Line(uint16_t address, size_t count, uint8_t * buffer, FILE * out);
```

File

idi.c (see page 373)

Description

This is function Hex_Dump_Line.

Body Source

```
1: int Hex_Dump_Line( uint16_t address, size_t count, uint8_t * buffer, FILE * out )
2: {
3:     size_t index;
4:     char str_temp[8];
5:     char str_ascii[20];
6:     char str_hex_list[64];
7:
8:
9:     if ( count > HEX_DUMP_BYTES_PER_LINE ) return -EC_HEX_DUMP_COUNT;
10:
11:    sprintf( str_hex_list, "%04X: ", ((int) address) );
12:    strcpy( str_ascii, " " );
13:    for ( index = 0; index < count; index++ )
14:    {
15:        sprintf( str_temp, "%02X", buffer[index] );
16:        strcat( str_hex_list, str_temp );
17:
18:        if ( 0x07 == ( index & 0x07 ) ) strcat( str_hex_list, " " );
19:
20:        strcat( str_hex_list, " " );
21:
22:        if ( ( buffer[index] < ' ' ) || ( buffer[index] > '~' ) )
23:        {
24:            strcat( str_ascii, "." );
25:        }
26:        else
27:        {
28:            sprintf( str_temp, "%c", buffer[index] );
29:            strcat( str_ascii, str_temp );
30:        }
31:    }
32:
33:    index = strlen( str_hex_list );
34:
35:    count = 60 - index;
36:    while ( count > 0 )
37:    {
38:        strcat( str_hex_list, " " );
39:        count--;
40:    }
41:
42:    fprintf( out, "%s%s\n", str_hex_list, str_ascii );
```

```

43:     return SUCCESS;
44: }
```

3.2.1.150 IAI_AI_ID_Get Function

C++

```
int IAI_AI_ID_Get(int board_id, uint16_t * id);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Obtains the analog input component (or board ID in this case) ID number.

param[out] id The 16-bit ID number

Body Source

```

1: int IAI_AI_ID_Get( int board_id, uint16_t * id )
2: {
3:     uint8_t     lsb, msb;
4:
5:     IO_Read_U8( board_id, IAI_ID_LSB, &lsb );
6:     IO_Read_U8( board_id, IAI_ID_MSB, &msb );
7:     *id = ( ((uint16_t) msb) << 8 ) | ((uint16_t) lsb);
8: #if defined( ID_ALWAYS_REPORT_AS_GOOD )
9:     *id = ID_IAI;
10: #endif
11:    return SUCCESS;
12: }
```

3.2.1.151 IAI16_AI_Channel_Get Function

C++

```
int IAI16_AI_Channel_Get(int board_id, size_t channel, int * value);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: int IAI16_AI_Channel_Get( int board_id, size_t channel, int * value )
2: {
3:     int     location;
```

```

4: size_t row;
5: uint16_t scratch;
6: struct reg_definition * definition = ( struct reg_definition * )
board_definition[board_id].definition;
7:
8: row = REG_LOCATION_LOGICAL_GET( IAI_AIN0 ) + channel;
9: location = REG_LOCATION_SET( row, 0 );
10:
11: IO_Read_U16_Address_Increment( board_id, location, &scratch );
12:
13: *value = (int) scratch;
14: return SUCCESS;
15: }
```

3.2.1.152 IAI16_AI_Channel_Multiple_Get Function

C++

```
int IAI16_AI_Channel_Multiple_Get(int board_id, size_t channel_first, size_t channel_last,
size_t table_offset, int * table);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: int IAI16_AI_Channel_Multiple_Get( int board_id,
2:                                     size_t channel_first,
3:                                     size_t channel_last,
4:                                     size_t table_offset,
5:                                     int * table
6:                                     )
7: {
8:     int location;
9:     size_t row;
10:    size_t index;
11:    uint16_t scratch;
12:    struct reg_definition * definition = ( struct reg_definition * )
board_definition[board_id].definition;
13:
14:    index = 0;
15:    for ( channel = channel_first; channel <= channel_last; channel++ )
16:    {
17:        row = REG_LOCATION_LOGICAL_GET( IAI_AIN0 ) + channel;
18:        location = REG_LOCATION_SET( row, 0 );
19:        IO_Read_U16_Address_Increment( board_id, location, &scratch );
20:
21:        value[table_offset + index] = (int) scratch;
22:        index++;
23:    }
24:    return SUCCESS;
25: }
```

3.2.1.153 IDI_Bank_Name_To_Symbol Function

C++

```
static int IDI_Bank_Name_To_Symbol(int board_id, const char * name, IDI_BANK_ENUM * bank);
```

File

idi.c (see page 373)

Returns

a string that describes the selected bank.

Description

brief

param[in] bank the bank enumerated value written to the bank register.

Body Source

```
1: static int IDI_Bank_Name_To_Symbol( int board_id, const char * name, IDI_BANK_ENUM * bank )
2: {
3:     int index;
4:     const struct bank_info * info = board_definition[board_id].bank_info;
5:     index = 0;
6:     while ( NULL != info[index].name )
7:     {
8:         if ( 0 == strcmpi( info[index].name, name ) )
9:         {
10:             *bank = info[index].symbol;
11:             return SUCCESS;
12:         }
13:         index++;
14:     }
15:     return -EC_BANK;
16: }
```

3.2.1.154 IDI_Bank_Symbol_To_Name Function

C++

```
static const char * IDI_Bank_Symbol_To_Name(struct idi_bank_info * info, IDI_BANK_ENUM bank);
```

File

idi.c (see page 373)

Returns

a string that describes the selected bank.

Description

brief Outputs a human readable CSV to the desired output file or stdout.

param[in] bank the bank enumerated value written to the bank register.

Body Source

```

1: static const char * IDI_Bank_Symbol_To_Name( struct idi_bank_info * info, IDI_BANK_ENUM
bank )
2: {
3:     int index;
4:
5:     switch( bank )
6:     {
7:         case IDI_BANK_0:    index = 0; break;
8:         case IDI_BANK_1:    index = 1; break;
9:         case IDI_BANK_2:    index = 2; break;
10:        case IDI_BANK_3:   index = 3; break;
11:        case IDI_BANK_4:   index = 4; break;
12:        case IDI_BANK_5:   index = 5; break;
13:        case IDI_BANK_6:   index = 6; break;
14:        case IDI_BANK_7:   index = 7; break;
15:        case IDI_BANK_NONE: index = 8; break;
16:        case IDI_BANK_UNDEFINED: index = 9; break;
17:    }
18:    return info[index].name;
19: }
```

3.2.1.155 IDI_Dataset_Defaults Function

C++

```
void IDI_Dataset_Defaults(struct idi_dataset * ds);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: void IDI_Dataset_Defaults( struct idi_dataset * ds )
2: {
3:     int index;
4:
5:     printf( "Warning - using main IDI48 dataset defaults\n" );
6:
7:     memset( ds, 0, sizeof(struct idi_dataset) );
8:
9:     ds->base_address      = 0x0110;
10:
11:
12:    for ( index = 0; index < REGS_INIT_QTY; index++ )
13:    {
14:        ds->reg_init[index].used      = false;
15:        ds->reg_init[index].address   = 0;
16:        ds->reg_init[index].location = IDI_UNDEFINED;
17:        ds->reg_init[index].value     = 0x00;
18:    }
19: }
```

3.2.1.156 IDI_DIN_Channel_Get Function

C++

```
int IDI_DIN_Channel_Get(int board_id, size_t channel, BOOL * value);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Obtains and reports a single digital input channel.

param[in] channel channel to be read out. param[out] value Pointer to the boolean value to be set based on the digital input value

Body Source

```
1: int IDI_DIN_Channel_Get( int board_id, size_t channel, BOOL * value )
2: {
3:     size_t group;
4:     size_t bit;
5:     uint8_t reg_value;
6:
7:     group = channel >> IDI_DIN_SHIFT_RIGHT;
8:     bit   = channel - group * IDI_DIN_GROUP_SIZE;
9:
10:    IO_Read_U8( board_id, __LINE__, IDI_DI_GROUP0 + group , &reg_value );
11:
12:    if ( 0 != ( reg_value & ( 0x01 << bit ) ) ) *value = true;
13:    else                                *value = false;
14:
15:    return SUCCESS;
16: }
```

3.2.1.157 IDI_DIN_Group_Get Function

C++

```
int IDI_DIN_Group_Get(int board_id, size_t group, uint8_t * value);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Reads the selected digital input port (8-bits).

param[in] group the group, range is 0 to 5. param[out] value pointer to the destination for the data read out

Body Source

```

1: int IDI_DIN_Group_Get( int board_id, size_t group, uint8_t * value )
2: {
3:   IO_Read_U8( board_id, __LINE__, IDI_DI_GROUP0 + group , value );
4:   return SUCCESS;
5: }
```

3.2.1.158 IDI_DIN_ID_Get Function

C++

```
int IDI_DIN_ID_Get(int board_id, uint16_t * id);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Obtains the digital input component (or board ID in this case) ID number.

param[out] id The 16-bit ID number

Body Source

```

1: int IDI_DIN_ID_Get( int board_id, uint16_t * id )
2: {
3:   uint8_t     lsb, msb;
4:   struct board_dataset * dataset;
5:
6:   dataset = (struct board_dataset *) board_definition[board_id].dataset;
7:
8:   if ( MODE_JUMPERS_0 == dataset->mode_jumpers )
9:   {
10:    return -EC_MODE_LEGACY;
11:   }
12:
13:   IO_Read_U8( board_id, __LINE__, IDI_ID_LSB, &lsb );
14:   IO_Read_U8( board_id, __LINE__, IDI_ID_MSB, &msb );
15:   *id = ( ((uint16_t) msb) << 8 ) | ((uint16_t) lsb);
16:
17:
18: #if defined( ID_ALWAYS_REPORT_AS_GOOD )
19:   *id = ID_DIN;
20: #endif
21:   return SUCCESS;
22: }
```

3.2.1.159 IDI_DIN_IsNotPresent Function

C++

```
BOOL IDI_DIN_IsNotPresent( int board_id );
```

File

idi.c (see page 373)

Returns

A zero is returned if present, otherwise a 1 is returned.

Description

brief Determines if the DIN component and/or board is present. Returns true if not present (i.e. error).

Body Source

```
1: BOOL IDI_DIN_IsNotPresent( int board_id )
2: {
3:     uint16_t id;
4:     IDI_DIN_ID_Get( board_id, &id );
5:     if (ID_DIN == id) return 0;
6:     return 1;
7: }
```

3.2.1.160 IDI_DIN_Pending_Clear Function

C++

```
int IDI_DIN_Pending_Clear( int board_id, size_t channel );
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

param[in] channel channel to be cleared of its pending status. param[out] value pointer to the destination for the data read out

Body Source

```
1: int IDI_DIN_Pending_Clear( int board_id, size_t channel )
2: {
3:     size_t group;
4:     size_t bit;
5:     uint8_t reg_value;
```

```

6:
7:   group = channel >> IDI_DIN_SHIFT_RIGHT;
8:   bit    = channel - group * IDI_DIN_GROUP_SIZE;
9:
10:  if ( group >= IDI_DIN_GROUP_QTY ) return -EC_CHANNEL;
11:
12:  reg_value = (uint8_t)( 0x01 << bit );
13:  IO_Write_U8( board_id, __LINE__, IDI_CLEAR_GROUP0 + group , reg_value );
14:  return SUCCESS;
15: }

```

3.2.1.161 IDI_Help Function

C++

```
static void IDI_Help(size_t * line, FILE * out);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static void IDI_Help( size_t * line, FILE * out )
2: {
3:   fprintf( out, "IDI48 board supported\n" );
4:   fprintf( out, " Examples:\n" );
5:   fprintf( out, "   idi dig0      <-- reports the digital input group 0 port.\n" );
6:   fprintf( out, "   idi loop dig0  <-- loops until key pressed\n" );
7:   fprintf( out, "   idi irq loop dig0 <-- loops while counting interrupts or until igtqy
reached\n" );
8:   fprintf( out, "   idi irq dig0    <-- does command once and uses interrupts for short
time\n" );
9:   *line += 7;
10: }

```

3.2.1.162 IDI_Initialization_Data_Structure_Load Function

C++

```
static void IDI_Initialization_Data_Structure_Load(struct idi_dataset * dataset);
```

File

idi.c (see page 373)

Description

brief

Body Source

```
1: static void IDI_Initialization_Data_Structure_Load( struct idi_dataset * dataset )
```

```

2:  {
3:    FILE * fd;
4:    size_t file_size;
5:
6:    fd = fopen( IDI_BOARD_INIT_FILE_NAME, "rb" );
7:    if ( NULL == fd )
8:    {
9:      IDI_Dataset_Defaults( dataset );
10:     #if defined( MAIN_DEBUG )
11:       printf( "IDI_Initialization_Data_Structure_Load - fd null -> using defaults\n" );
12:     #endif
13:    }
14:   else
15:   {
16:     fseek( fd, 0L, SEEK_END );
17:     file_size = (size_t) ftell( fd );
18:     fseek( fd, 0L, SEEK_SET );
19:     if ( file_size != sizeof( struct idi_dataset ) )
20:     {
21:       IDI_Dataset_Defaults( dataset );
22:     #if defined( MAIN_DEBUG )
23:       printf( "IDI_Initialization_Data_Structure_Load - file_size=%u\n" );
24:     #if defined( __MSDOS__ )
25:       printf( "%u, sizeof(struct idi_dataset)=%u\n", file_size, sizeof( struct idi_dataset ) );
26:     #else
27:       printf( "%lu, sizeof(struct idi_dataset)=%lu\n", file_size, sizeof( struct idi_dataset ) );
28:     #endif
29:     #endif
30:   }
31:   else
32:   {
33:     fread( dataset, sizeof( struct idi_dataset ), 1, fd );
34:     #if defined( MAIN_DEBUG )
35:       printf( "IDI_Initialization_Data_Structure_Load - file_size=%u\n" );
36:     #if defined( __MSDOS__ )
37:       printf( "%u\n", file_size );
38:     #else
39:       printf( "%lu\n", file_size );
40:     #endif
41:     #endif
42:   }
43:   fclose( fd );
44: }
45: dataset->svn_revision_string      = svn_revision_string;
46: dataset->irq_handler_active      = false;
47: dataset->irq.PleaseInstallHandlerRequest = false;
48: dataset->irq_count              = 0;
49: dataset->irq_count_previous     = 0;
50: dataset->spi_is_present        = false;
51:
52: dataset->bank_previous          = IDI_BANK_UNDEFINED;
53: dataset->quit_application       = false;
54: dataset->board_id               = ID_IDI48;
55: }
```

3.2.1.163 IDI_Initialization_Register Function

C++

```
static void IDI_Initialization_Register(int board_id, struct idi_dataset * dataset);
```

Copyright ©2015 by [Apex Embedded Systems](#). All rights reserved.

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static void IDI_Initialization_Register( int board_id, struct idi_dataset * dataset )
2: {
3:     int index;
4:
5:     for ( index = 0; index < REGS_INIT_QTY; index++ )
6:     {
7:         if ( dataset->reg_init[index].used )
8:         {
9:             if ( IDI_UNDEFINED != dataset->reg_init[index].location )
10:             {
11:                 IO_Write_U8( board_id, __LINE__, dataset->reg_init[index].location,
dataset->reg_init[index].value );
12:             }
13:         else
14:         {
15:             IO_Write_U8_Port( (struct board_dataset *) dataset,
16:                               __LINE__,
17:                               dataset->reg_init[index].address,
18:                               dataset->reg_init[index].value
19:                           );
20:         }
21:     }
22: }
23: }
```

3.2.1.164 IDI_Main_Loop_Testing Function

C++

```
int IDI_Main_Loop_Testing(struct idi_dataset * dataset);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: int IDI_Main_Loop_Testing( struct idi_dataset * dataset )
2: {
3:     if ( dataset->irq_handler_active )
4:     {
5:         if ( dataset->irq_count != dataset->irq_count_previous )
6:         {
7: 
```

```

8:     dataset->irq_count_previous = dataset->irq_count;
9:
10:    #if defined( __MSDOS__ )
11:        printf( " --> irq %u: count = %d\n", dataset->irq_number, dataset->irq_count );
12:    #else
13:        printf( " --> irq %u: count = %lu\n", dataset->irq_number, dataset->irq_count );
14:    #endif
15:    if ( dataset->irq_count_previous >= dataset->irq_quantity )
16:    {
17:        return -1;
18:    }
19: }
20: }
21: return 0;
22: }
```

3.2.1.165 IDI_Register_Report_CSV Function

C++

`static int IDI_Register_Report_CSV(const struct idi_reg_definition * table, FILE * out);`

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int IDI_Register_Report_CSV( const struct idi_reg_definition * table, FILE * out )
2: {
3:     int index = 0;
4:
5:     fprintf( out, "\"acronym\",\"symbol\",\"bank\",\"direction\",\"physical_offset\"\n" );
6:     do
7:     {
8:         fprintf( out, "\"%s\",", table[index].acronym );
9:         fprintf( out, "\"%s\",", table[index].symbol_name );
10:
11:        switch( table[index].bank )
12:        {
13:            case IDI_BANK_0:   fprintf( out, "IDI_BANK_0" );   break;
14:            case IDI_BANK_1:   fprintf( out, "IDI_BANK_1" );   break;
15:            case IDI_BANK_2:   fprintf( out, "IDI_BANK_2" );   break;
16:            case IDI_BANK_3:   fprintf( out, "IDI_BANK_3" );   break;
17:            case IDI_BANK_4:   fprintf( out, "IDI_BANK_4" );   break;
18:            case IDI_BANK_5:   fprintf( out, "IDI_BANK_5" );   break;
19:            case IDI_BANK_6:   fprintf( out, "IDI_BANK_6" );   break;
20:            case IDI_BANK_7:   fprintf( out, "IDI_BANK_7" );   break;
21:            case IDI_BANK_NONE:   fprintf( out, "IDI_BANK_NONE" );   break;
22:            case IDI_BANK_UNDEFINED:   fprintf( out, "IDI_BANK_UNDEFINED" );   break;
23:        }
24:        fprintf( out, "," );
25:
26:        switch( table[index].direction )
27:        {
28:            case REG_DIR_NONE:   fprintf( out, "REG_DIR_NONE" );   break;
29:            case REG_DIR_READ:   fprintf( out, "REG_DIR_READ" );   break;
```

```

30:     case REG_DIR_WRITE:   fprintf( out, "REG_DIR_WRITE" );  break;
31:     case REG_DIR_READ_WRITE: fprintf( out, "REG_DIR_READ_WRITE" );  break;
32:   }
33:   fprintf( out, "," );
34:
35:   fprintf( out, "\"%d\"", table[index].physical_offset );
36:   fprintf( out, "\n\r" );
37:   index++;
38: } while ( table[index].direction != REG_DIR_NONE );
39:
40: return SUCCESS;
41: }
```

3.2.1.166 IDI_Termination Function

C++

```
static int IDI_Termination(struct idi_dataset * dataset);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int IDI_Termination( struct idi_dataset * dataset )
2: {
3:   FILE * fd;
4:
5: #if defined( MAIN_DEBUG )
6: printf( "Main:  IDI_Termination\n" );
7: #endif
8:
9: #if defined( __MSDOS__ )
10: IOKern_Resource_Termination();
11: #endif
12: if ( dataset->irq_handler_active )
13: {
14: #if defined( __MSDOS__ )
15: printf( " --> irq %u: count = %d\n", dataset->irq_number, dataset->irq_count );
16: #else
17:   printf( " --> irq %u: count = %lu\n", dataset->irq_number, dataset->irq_count );
18: #endif
19: }
20: dataset->irq_handler_active = false;
21: dataset->irq_please_install_handler_request = false;
22:
23:
24: fd = fopen( IDI_BOARD_INIT_FILE_NAME, "wb" );
25: if ( NULL == fd )
26: {
27:
28: #if defined( MAIN_DEBUG )
29:   printf("IDI_Termination: fopen() error: %d (%s)\n", errno, strerror(errno) );
30: #endif
31:   return -EC_INIT_FILE;
32: }
```

```

33: else
34: {
35: #if defined( MAIN_DEBUG )
36: printf( "IDI_Termination: sizeof( struct idi_dataset ) = " );
37: # if defined( __MSDOS__ )
38:     printf( "%u\n", sizeof( struct idi_dataset ) );
39: # else
40:     printf( "%lu\n", sizeof( struct idi_dataset ) );
41: # endif
42: #endif
43:
44:     fwrite( dataset, sizeof( struct idi_dataset ), 1, fd );
45:     fclose( fd );
46: }
47: return SUCCESS;
48: }
```

3.2.1.167 IDI48_DIN_ID_Port_Scan Function

C++

```
static int IDI48_DIN_ID_Port_Scan(const struct port_list * pl, size_t * row, int * port);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int IDI48_DIN_ID_Port_Scan( const struct port_list * pl, size_t * row, int * port )
2: {
3:     size_t      row_index;
4:     int         pa;
5:     int         pi_start, pi_end;
6:     int         count;
7:     BOOL        not_done;
8:     uint16_t    id;
9:     uint8_t     lsb, msb;
10:
11: #if defined( __MSDOS__ )
12: #else
13:     return -EC_NOT_IMPLEMENTED;
14: #endif
15:
16:     not_done = true;
17:     row_index = 0;
18:     do
19:     {
20:         if ( ( 0 == pl[row_index].address_start ) && ( 0 == pl[row_index].address_end ) )
21:         {
22:             not_done = false;
23:         }
```

```

24: else
25: {
26:     printf ( "." );
27:     count = pl[row_index].address_end - pl[row_index].address_start + 1;
28:     if ( count > 1 )
29:     {
30:         pi_start = pl[row_index].address_start      ;
31:         pi_end   = pl[row_index].address_end        + 1 ;
32:         if ( 1 == ( pi_start & 1 ) )
33:         {
34:             pi_start++;
35:             count--;
36:         }
37:
38:         for ( pa = pi_start; pa < pi_end; pa = pa + 2 )
39:         {
40: #if defined( __MSDOS__ )
41:             lsb = inportb( ( pa + 0 ) );
42:             msb = inportb( ( pa + 1 ) );
43: #endif
44:             id  = ( ((uint16_t) msb) << 8 ) | ((uint16_t) lsb);
45:
46:             if ( ID_DIN == id )
47:             {
48:                 *row  = row_index;
49:                 *port = pa;
50:                 return SUCCESS;
51:             }
52:         }
53:     }
54:     row_index++;
55: }
56: } while ( true == not_done );
57:
58: return -EC_NOT_FOUND;
59: }
```

3.2.1.168 IDO_Bank_Name_To_Symbol Function

C++

```
static int IDO_Bank_Name_To_Symbol(int board_id, const char * name, IDO_BANK_ENUM * bank);
```

File

idi.c (see page 373)

Description

This is function IDO_Bank_Name_To_Symbol.

Body Source

```

1: static int IDO_Bank_Name_To_Symbol( int board_id, const char * name, IDO_BANK_ENUM * bank )
2: {
3:     int index;
4:     const struct bank_info * info = board_definition[board_id].bank_info;
5:     index = 0;
6:     while ( NULL != info[index].name )
7:     {
8:         if ( 0 == strcmpi( info[index].name, name ) )
```

```

9:  {
10:    *bank = info[index].symbol;
11:    return SUCCESS;
12:  }
13:  index++;
14: }
15: return -EC_BANK;
16: }
```

3.2.1.169 IDO_Bank_Symbol_To_Name Function

C++

```
static const char * IDO_Bank_Symbol_To_Name(struct ido_bank_info * info, IDO_BANK_ENUM bank);
```

File

idi.c (see page 373)

Description

This is function IDO_Bank_Symbol_To_Name.

Body Source

```

1: static const char * IDO_Bank_Symbol_To_Name( struct ido_bank_info * info, IDO_BANK_ENUM
bank )
2: {
3:   int index;
4:
5:   switch( bank )
6:   {
7:     case IDO_BANK_0:   index = 0; break;
8:     case IDO_BANK_1:   index = 1; break;
9:     case IDO_BANK_2:   index = 2; break;
10:    case IDO_BANK_3:  index = 3; break;
11:    case IDO_BANK_4:  index = 4; break;
12:    case IDO_BANK_5:  index = 5; break;
13:    case IDO_BANK_6:  index = 6; break;
14:    case IDO_BANK_7:  index = 7; break;
15:    case IDO_BANK_NONE: index = 8; break;
16:    case IDO_BANK_UNDEFINED: index = 9; break;
17:   }
18:   return info[index].name;
19: }
```

3.2.1.170 IDO_Dataset_Defaults Function

C++

```
void IDO_Dataset_Defaults(struct ido_dataset * ds);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: void IDO_Dataset_Defaults( struct ido_dataset * ds )
2: {
3:     int index;
4:
5:     printf( "Warning - using main IDO48 dataset defaults\n" );
6:
7:     memset( ds, 0, sizeof(struct ido_dataset) );
8:
9:     ds->base_address      = 0x0130;
10:
11:
12:    for ( index = 0; index < REGS_INIT_QTY; index++ )
13:    {
14:        ds->reg_init[index].used      = false;
15:        ds->reg_init[index].address   = 0;
16:        ds->reg_init[index].location = IDO_UNDEFINED;
17:        ds->reg_init[index].value    = 0x00;
18:    }
19: }
```

3.2.1.171 IDO_DO_Channel_Get Function

C++

```
int IDO_DO_Channel_Get(int board_id, size_t channel, BOOL * value);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Obtains and reports a single digital output channel.

param[in] channel channel to be read out. param[out] value Pointer to the boolean value to be set based on the digital input value

Body Source

```

1: int IDO_DO_Channel_Get( int board_id, size_t channel, BOOL * value )
2: {
3:     size_t group;
4:     size_t bit;
5:     uint8_t reg_value;
6:
7:     group = channel >> IDO_DO_SHIFT_RIGHT;
8:     bit   = channel - group * IDO_DO_GROUP_SIZE;
9:
```

```

10: IO_Read_U8( board_id, __LINE__, IDO_DO_GROUP0 + group , &reg_value );
11:
12: if ( 0 != ( reg_value & ( 0x01 << bit ) ) ) *value = true;
13: else                                *value = false;
14:
15: return SUCCESS;
16: }
```

3.2.1.172 IDO_DO_Channel_Set Function

C++

```
int IDO_DO_Channel_Set(int board_id, size_t channel, BOOL value);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Sets a single digital output channel.

param[in] channel to be written. param[in] value

Body Source

```

1: int IDO_DO_Channel_Set( int board_id, size_t channel, BOOL value )
2: {
3:     size_t group;
4:     size_t bit;
5:     uint8_t reg_value;
6:     uint8_t mask;
7:
8:     group = channel >> IDO_DO_SHIFT_RIGHT;
9:     bit   = channel - group * IDO_DO_GROUP_SIZE;
10:
11:    IO_Read_U8( board_id, __LINE__, IDO_DO_GROUP0 + group , &reg_value );
12:    mask = 0x01 << bit;
13:    if ( true == value ) reg_value |= mask;
14:    else      reg_value &= ~mask;
15:
16:    IO_Write_U8( board_id, __LINE__, IDO_DO_GROUP0 + group , reg_value );
17:    return SUCCESS;
18: }
```

3.2.1.173 IDO_DO_Group_Get Function

C++

```
int IDO_DO_Group_Get(int board_id, size_t group, uint8_t * value);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Reads the selected digital output port (8-bits).

param[in] group the group, range is 0 to 5. param[out] value pointer to the destination for the data read out

Body Source

```
1: int IDO_DO_Group_Get( int board_id, size_t group, uint8_t * value )
2: {
3:   IO_Read_U8( board_id, __LINE__, IDO_DO_GROUP0 + group , value );
4:   return SUCCESS;
5: }
```

3.2.1.174 IDO_DO_Group_Set Function

C++

```
int IDO_DO_Group_Set(int board_id, size_t group, uint8_t value);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Reads the selected digital output port (8-bits).

param[in] group the group, range is 0 to 5. param[in] value pointer to the destination for the data read out

Body Source

```
1: int IDO_DO_Group_Set( int board_id, size_t group, uint8_t value )
2: {
3:   IO_Write_U8( board_id, __LINE__, IDO_DO_GROUP0 + group , value );
4:   return SUCCESS;
5: }
```

3.2.1.175 IDO_DOUT_ID_Get Function

C++

```
int IDO_DOUT_ID_Get(int board_id, uint16_t * id);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Obtains the digital output component (or board ID in this case) ID number.

param[out] id The 16-bit ID number

Body Source

```

1: int IDO_DOUT_ID_Get( int board_id, uint16_t * id )
2: {
3:     uint8_t lsb, msb;
4:     struct board_dataset * dataset;
5:     dataset = (struct board_dataset *) board_definition[board_id].dataset;
6:
7:     if ( MODE_JUMPERS_0 == dataset->mode_jumpers )
8:     {
9:         return -EC_MODE_LEGACY;
10:    }
11:
12:    IO_Read_U8( board_id, __LINE__, IDO_ID_LSB, &lsb );
13:    IO_Read_U8( board_id, __LINE__, IDO_ID_MSB, &msb );
14:    *id = ( ((uint16_t) msb) << 8 ) | ((uint16_t) lsb);
15: #if defined( ID_ALWAYS_REPORT_AS_GOOD )
16:    *id = ID_DOUT;
17: #endif
18:    return SUCCESS;
19: }
```

3.2.1.176 IDO_DOUT_IsNotPresent Function

C++

```
BOOL IDO_DOUT_IsNotPresent( int board_id );
```

File

idi.c (see page 373)

Returns

A zero is returned if present, otherwise a 1 is returned.

Description

brief Determines if the DIN component and/or board is present. Returns true if not present (i.e. error).

Body Source

```

1: BOOL IDO_DOUT_IsNotPresent( int board_id )
2: {
3:     uint16_t id;
4:     IDO_DOUT_ID_Get( board_id, &id );
5:     if ( ID_DOUT == id ) return 0;
6:     return 1;
```

```
7: }
```

3.2.1.177 IDO_Help Function

C++

```
static void IDO_Help(size_t * line, FILE * out);
```

File

idi.c (see page 373)

Description

brief

Body Source

```
1: static void IDO_Help( size_t * line, FILE * out )
2: {
3:   fprintf( out, "IDO48 board supported\n" );
4:   fprintf( out, " Examples:\n" );
5:   fprintf( out, "    ido dog0          <-- reports the digital output group 0 port.\n" );
6:   fprintf( out, "    ido loop dog0      <-- loops until key pressed\n" );
7:   *line += 5;
8: }
```

3.2.1.178 IDO_Initialization_Data_Structure_Load Function

C++

```
static void IDO_Initialization_Data_Structure_Load(struct ido_dataset * dataset);
```

File

idi.c (see page 373)

Description

brief

Body Source

```
1: static void IDO_Initialization_Data_Structure_Load( struct ido_dataset * dataset )
2: {
3:   FILE * fd;
4:   size_t file_size;
5:
6:   fd = fopen( IDO_BOARD_INIT_FILE_NAME, "rb" );
7:   if ( NULL == fd )
8:   {
9:     IDO_Dataset_Defaults( dataset );
10: #if defined( MAIN_DEBUG )
11:   printf( "IDO_Initialization_Data_Structure_Load - fd null -> using defaults\n" );
12: #endif
13:   }
14: else
```

```

15: {
16:     fseek( fd, 0L, SEEK_END );
17:     file_size = (size_t) ftell( fd );
18:     fseek( fd, 0L, SEEK_SET );
19:     if ( file_size != sizeof( struct ido_dataset ) )
20:     {
21:         IDO_Dataset_Defaults( dataset );
22: #if defined( MAIN_DEBUG )
23:         printf( "IDO_Initialization_Data_Structure_Load - file_size=%u\n" );
24: #if defined( __MSDOS__ )
25:         printf( "%u, sizeof(struct ido_dataset)=%u\n", file_size, sizeof( struct ido_dataset ) );
26: #else
27:         printf( "%lu, sizeof(struct ido_dataset)=%lu\n", file_size, sizeof( struct ido_dataset ) );
28: #endif
29: #endif
30:     }
31:     else
32:     {
33:         fread( dataset, sizeof( struct ido_dataset ), 1, fd );
34: #if defined( MAIN_DEBUG )
35:         printf( "IDO_Initialization_Data_Structure_Load - file_size=%u\n" );
36: #if defined( __MSDOS__ )
37:         printf( "%u\n", file_size );
38: #else
39:         printf( "%lu\n", file_size );
40: #endif
41: #endif
42:     }
43:     fclose( fd );
44: }
45: dataset->svn_revision_string      = svn_revision_string;
46: dataset->irq_handler_active      = false;
47: dataset->irq.PleaseInstallHandlerRequest = false;
48: dataset->irq_count               = 0;
49: dataset->irq_count_previous     = 0;
50: dataset->spi_is_present        = false;
51:
52: dataset->bank_previous          = IDO_BANK_UNDEFINED;
53: dataset->quit_application       = false;
54: dataset->board_id               = ID_IDO48;
55: }
```

3.2.1.179 IDO_Initialization_Register Function

C++

```
static void IDO_Initialization_Register(int board_id, struct ido_dataset * dataset);
```

File

idi.c (see page 373)

Description

brief

Body Source

```
1: static void IDO_Initialization_Register( int board_id, struct ido_dataset * dataset )
```

Copyright ©2015 by [Apex Embedded Systems](#). All rights reserved.

```

2: {
3:     int index;
4:
5:     for ( index = 0; index < REGS_INIT_QTY; index++ )
6:     {
7:         if ( dataset->reg_init[index].used )
8:         {
9:             if ( IDO_UNDEFINED != dataset->reg_init[index].location )
10:             {
11:                 IO_Write_U8( board_id, __LINE__, dataset->reg_init[index].location,
dataset->reg_init[index].value );
12:             }
13:         else
14:         {
15:             IO_Write_U8_Port( (struct board_dataset *) dataset,
16:                               __LINE__,
17:                               dataset->reg_init[index].address,
18:                               dataset->reg_init[index].value
19:                           );
20:         }
21:     }
22: }
23: }
```

3.2.1.180 IDO_Register_Report_CSV Function

C++

```
static int IDO_Register_Report_CSV(const struct ido_reg_definition * table, FILE * out);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int IDO_Register_Report_CSV( const struct ido_reg_definition * table, FILE * out )
2: {
3:     int index = 0;
4:
5:     fprintf( out, "\"acronym\",\"symbol\",\"bank\",\"direction\",\"physical_offset\"\n" );
6:     do
7:     {
8:         fprintf( out, "%s,", table[index].acronym );
9:         fprintf( out, "%s,", table[index].symbol_name );
10:
11:        switch( table[index].bank )
12:        {
13:            case IDO_BANK_0:   fprintf( out, "IDO_BANK_0" ); break;
14:            case IDO_BANK_1:   fprintf( out, "IDO_BANK_1" ); break;
15:            case IDO_BANK_2:   fprintf( out, "IDO_BANK_2" ); break;
16:            case IDO_BANK_3:   fprintf( out, "IDO_BANK_3" ); break;
17:            case IDO_BANK_4:   fprintf( out, "IDO_BANK_4" ); break;
18:            case IDO_BANK_5:   fprintf( out, "IDO_BANK_5" ); break;
19:            case IDO_BANK_6:   fprintf( out, "IDO_BANK_6" ); break;
20:            case IDO_BANK_7:   fprintf( out, "IDO_BANK_7" ); break;
21:            case IDO_BANK_NONE: fprintf( out, "IDO_BANK_NONE" ); break;
```

```

22:   case IDO_BANK_UNDEFINED: fprintf( out, "IDO_BANK_UNDEFINED" ); break;
23: }
24: fprintf( out, "," );
25:
26: switch( table[index].direction )
27: {
28:   case REG_DIR_NONE:   fprintf( out, "REG_DIR_NONE" );   break;
29:   case REG_DIR_READ:   fprintf( out, "REG_DIR_READ" );   break;
30:   case REG_DIR_WRITE:  fprintf( out, "REG_DIR_WRITE" );  break;
31:   case REG_DIR_READ_WRITE: fprintf( out, "REG_DIR_READ_WRITE" ); break;
32: }
33: fprintf( out, "," );
34:
35: fprintf( out, "\"%d\"", table[index].physical_offset );
36: fprintf( out, "\n\r" );
37: index++;
38: } while ( table[index].direction != REG_DIR_NONE );
39:
40: return SUCCESS;
41: }
```

3.2.1.181 IDO_Termination Function

C++

```
static int IDO_Termination(struct ido_dataset * dataset);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int IDO_Termination( struct ido_dataset * dataset )
2: {
3:   FILE * fd;
4:
5: #if defined( MAIN_DEBUG )
6: printf( "Main: IDO_Termination\n" );
7: #endif
8:
9:
10: fd = fopen( IDO_BOARD_INIT_FILE_NAME, "wb" );
11: if ( NULL == fd )
12: {
13: #if defined( MAIN_DEBUG )
14:   printf("IDO_Termination: fopen() error: %d (%s)\n", errno, strerror(errno) );
15: #endif
16:   return -EC_INIT_FILE;
17: }
18: else
19: {
20: #if defined( MAIN_DEBUG )
21: printf( "IDO_Termination: sizeof( struct ido_dataset ) = " );
22: # if defined( __MSDOS__ )
23:   printf( "%u\n", sizeof( struct ido_dataset ) );
24: # else
```

```

25:     printf( "%lu\n", sizeof( struct ido_dataset ) );
26: # endif
27: #endif
28:     fwrite( dataset, sizeof( struct ido_dataset ), 1, fd );
29:     fclose( fd );
30: }
31: return SUCCESS;
32: }
```

3.2.1.182 IDO48_IDI48_Loopback_Test Function

C++

```
static int IDO48_IDI48_Loopback_Test(int di_id, int do_id, uint8_t * di_list, uint8_t *
do_list, size_t count);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int IDO48_IDI48_Loopback_Test( int di_id, int do_id, uint8_t * di_list, uint8_t *
do_list, size_t count )
2: {
3:     int group;
4:     for( group = 0; group < count; group++ )
5:     {
6:         IDO_DO_Group_Get( do_id, group, &(do_list[group]) );
7:         IDI_DIN_Group_Get( di_id, group, &(di_list[group]) );
8:         if ( di_list[group] != do_list[group] ) return -group;
9:     }
10:    return SUCCESS;
11: }
```

3.2.1.183 Initialization Function

C++

```
int Initialization(int board_id);
```

File

idi.c (see page 373)

Returns

SUCCESS (0) if no errors encountered, otherwise errors are reported as a negative value.

Description

brief Runs upon application startup. It restores the idi_dataset (see page 302) data structure or if the file cannot be found it will simply initialize those parameters to default values.

Body Source

```

1: int Initialization( int board_id )
2: {
3:     struct board_dataset * dataset;
4:     (void) board_id;
5:
6:     dataset = (struct board_dataset *) board_definition[ID_IDI48].dataset;
7:     dataset->quit_application = false;
8:     IDI_Initialization_Data_Structure_Load( (struct idi_dataset *) dataset );
9:     dataset = (struct board_dataset *) board_definition[ID_IDO48].dataset;
10:    dataset->quit_application = false;
11:    IDO_Initialization_Data_Structure_Load( (struct ido_dataset *) dataset );
12:
13: #if defined( __MSDOS__ )
14:     IOKern_Resource_Initialization();
15: #endif
16:     dataset = (struct board_dataset *) board_definition[ID_IDI48].dataset;
17:     IDI_Initialization_Register( ID_IDI48, (struct idi_dataset *) dataset );
18:     dataset = (struct board_dataset *) board_definition[ID_IDO48].dataset;
19:     IDO_Initialization_Register( ID_IDO48, (struct ido_dataset *) dataset );
20:
21:
22:     signal( SIGINT, Signal_Handler );
23:
24:     IO_Trace_Initialize( ID_IDI48 );
25:     IO_Trace_Initialize( ID_IDO48 );
26:
27: #if(0)
28:     struct board_dataset * dataset;
29:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
30:
31:
32:     switch( board_id )
33:     {
34:         case ID_IDI48:
35:             IDI_Initialization_Data_Structure_Load( (struct idi_dataset *) dataset );
36:             break;
37:         case ID_IDO48:
38:             IDO_Initialization_Data_Structure_Load( (struct ido_dataset *) dataset );
39:             break;
40:         default:
41:             return -EC_BOARD_TYPE;
42:     }
43: #if defined( __MSDOS__ )
44:     IOKern_Resource_Initialization();
45: #endif
46:
47:     switch( board_id )
48:     {
49:         case ID_IDI48:
50:             IDI_Initialization_Register( board_id, (struct idi_dataset *) dataset );
51:             break;
52:         case ID_IDO48:
53:             IDO_Initialization_Register( board_id, (struct ido_dataset *) dataset );
54:             break;
55:         default:

```

```

56:     return -EC_BOARD_TYPE;
57: }
58:
59:
60: dataset->quit_application = false;
61: signal( SIGINT, Signal_Handler );
62: #endif
63:     return SUCCESS;
64: }
```

3.2.1.184 IO_Direction_IsNotValid Function

C++

```
static BOOL IO_Direction_IsNotValid(IDI_REG_ENUM location, REG_DIR_ENUM direction);
```

File

idi.c (see page 373)

Returns

A false is returned if the direction is valid, otherwise a true is returned

Description

brief Looks up in the register definitions list for the ports possible read/write directions.

param[in] location the enumerated register symbol. The enumerated symbol is composed of offset and bank information used to determine the final address information. param[in] direction The desired direction that is to run

Body Source

```

1: static BOOL IO_Direction_IsNotValid( IDI_REG_ENUM location, REG_DIR_ENUM direction )
2: {
3:     int index;
4:     index = REG_LOCATION_LOGICAL_GET( location );
5:     if ( direction == (idi_definitions[index].direction & direction) ) return false;
6:     return true;
7: }
```

3.2.1.185 IO_Get_Symbol_Name Function

C++

```
static char * IO_Get_Symbol_Name(IDI_REG_ENUM location);
```

File

idi.c (see page 373)

Returns

a human readable string of the symbol.

Description

brief Translates a register enumerated symbol into a string that is the same as the enumerated symbol used throughout this code base.

param[in] location The enumerated symbol representing the register.

Body Source

```
1: static char * IO_Get_Symbol_Name( IDI_REG_ENUM location )
2: {
3:     int index;
4:     index = REG_LOCATION_LOGICAL_GET( location );
5:     return idi_definitions[index].symbol_name;
6: }
```

3.2.1.186 IO_Read_U16_Address_Fixed Function

C++

```
void IO_Read_U16_Address_Fixed(int board_id, size_t line, int location, uint16_t * value);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Reads uint16_t (see page 321) from I/O ports in a uint8_t (see page 321) succession to the same address location. Macros are used to guide the target implementation. In this case, bus width (which we typically refer to the port width, which is different than register width) is assumed to be byte (uint8_t (see page 321)) wide.

param[in] location the enumerated register symbol. The enumerated symbol is composed of offset and bank information used to determine the final address information. param[in] value The pointer to the destination of the read uint16_t (see page 321) value.

Body Source

```
1: void IO_Read_U16_Address_Fixed( int board_id, size_t line, int location, uint16_t * value )
2: {
3:
4:     uint8_t lsb, msb;
5:     IO_Read_U8( board_id, line, (int)((int)location + 0), &lsb );
6:     IO_Read_U8( board_id, line, (int)((int)location + 0), &msb );
7:     *value = ( ((uint16_t) msb) << 8 ) | ( ((uint16_t) lsb) & 0xFF );
8: }
```

3.2.1.187 IO_Read_U16_Address_Increment Function

C++

```
void IO_Read_U16_Address_Increment(int board_id, size_t line, int location, uint16_t * value);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Reads uint16_t (see page 321) from I/O ports in a uint8_t (see page 321) succession incrementing the offset address. Macros are used to guide the target implementation. In this case, bus width (which we typically refer to the port width, which is different than register width) is assumed to be byte (uint8_t (see page 321)) wide.

param[in] location the enumerated register symbol. The enumerated symbol is composed of offset and bank information used to determine the final address information. param[in] value The pointer to the destination of the read uint16_t (see page 321) value.

Body Source

```
1: void IO_Read_U16_Address_Increment( int board_id, size_t line, int location, uint16_t *  
value )  
2: {  
3:  
4:     uint8_t lsb, msb;  
5:     IO_Read_U8( board_id, line, (int)((int) location + 0 ), &lsb );  
6:     IO_Read_U8( board_id, line, (int)((int) location + 1 ), &msb );  
7:     *value = ( ((uint16_t) msb) << 8 ) | ( ((uint16_t) lsb) & 0xFF );  
8: }
```

3.2.1.188 IO_Read_U8 Function

C++

```
int IO_Read_U8(int board_id, size_t line, int location, uint8_t * value);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Reads uint8_t (see page 321) from I/O port. Macros are used to guide the target implementation.

param[in] location the enumerated register symbol. The enumerated symbol is composed of offset and bank information used to determine the final address information. param[in] value The pointer to the destination of the read uint8_t (see page 321) value.

Body Source

```

1: int IO_Read_U8( int board_id, size_t line, int location, uint8_t * value )
2: {
3:     int error_code;
4:     int index;
5:     int offset;
6:     int channel;
7:     int address;
8:     int bank;
9:
10:    struct reg_definition * definition = ( struct reg_definition * )
board_definition[board_id].definition;
11:    struct board_dataset * dataset      = ( struct board_dataset * )
board_definition[board_id].dataset;
12:
13:    error_code = SUCCESS;
14:    index     = REG_LOCATION_LOGICAL_GET( location );
15: #if defined( IDI_IO_DIRECTION_TEST )
16:    if ( !( REG_DIR_READ == ( definition[index].direction & REG_DIR_READ ) ) )
17:    {
18:        printf( "IO_Read_U8: %s, error in direction\n", definition[index].symbol_name );
19:        error_code = -EC_DIRECTION;
20:        goto IO_Read_U8_Exit;
21:    }
22: #endif
23:    bank    = definition[index].bank;
24:    if ( ( BANK_NONE != bank ) && ( bank != dataset->bank_previous ) )
25:    {
26:        address = dataset->base_address + definition[REG_LOCATION_LOGICAL_GET(
board_definition[board_id].bank_register_symbol)].physical_offset;
27:        dataset->bank_previous = bank;
28:        IO_Write_U8_Port( dataset, __LINE__, address, (uint8_t) bank );
29:        IO_Trace_Log( board_id, line, IO_TRACE_READ,
board_definition[board_id].bank_register_symbol, error_code, (uint8_t) bank );
30:        if ( ( dataset->io_report ) || ( dataset->io_simulate ) )
31:        {
32:            printf( "IO_Read_U8 bank write: %s, address = 0x%04X, bank = %s (0x%02X)\n",
definition[index].symbol_name,
33:                   address,
34:                   Bank_Symbol_To_Name( board_id, bank ),
35:                   bank
36:             );
37:        }
38:    }
39:    channel = REG_LOCATION_CHANNEL_GET( location );
40:    offset   = definition[index].physical_offset;
41:    address  = dataset->base_address + offset + channel;
42:    IO_Read_U8_Port( dataset, __LINE__, address, value );
43:    IO_Read_U8_Exit:
44:    IO_Trace_Log( board_id, line, IO_TRACE_READ, location, error_code, *value );
45:    return SUCCESS;
46: }
```

3.2.1.189 IO_Read_U8_Port Function

C++

```
static int IO_Read_U8_Port(struct board_dataset * dataset, size_t line, int address, uint8_t * value);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int IO_Read_U8_Port( struct board_dataset * dataset, size_t line, int address,
2:     uint8_t * value )
3: {
4:     if ( NULL == dataset )
5:     {
6:         #if defined( __MSDOS__ )
7:             *value = inportb( address );
8:         printf( "IO_Read_U8_Port: address = 0x%04X, value=unknown\n", address );
9:     #endif
10:    }
11:    else
12:    {
13:        if ( !dataset->io_simulate )
14:        {
15:            #if defined( __MSDOS__ )
16:                *value = inportb( address );
17:            #endif
18:        }
19:        if ( ( dataset->io_report ) || ( dataset->io_simulate ) )
20:        {
21:            printf( "IO_Read_U8_Port: address = 0x%04X, ", address );
22:            #if defined( __MSDOS__ )
23:                printf( "value = 0x%02X\n", *value );
24:            #else
25:                printf( "value = unknown\n" );
26:            #endif
27:        }
28:    }
29:    return SUCCESS;
30: }
```

3.2.1.190 IO_Trace_Dump Function

C++

```
int IO_Trace_Dump(int board_id);
```

File

idi.c (see page 373)

Description

brief Dumps the trace buffer (see page 325) to a CSV file for further analysis.

Body Source

```

1: int IO_Trace_Dump( int board_id )
2: {
3:     int error_code;
4:     FILE * out;
5:     size_t count, index, row;
6:     struct io_trace * trace;
7:     struct reg_definition * definition = ( struct reg_definition * )
board_definition[board_id].definition;
8:     struct board_dataset * dataset = ( struct board_dataset * )
board_definition[board_id].dataset;
9:
10:    if ( false == global_io_trace_enable ) return SUCCESS;
11:
12:    if ( strlen(dataset->trace.filename) == 0 ) out = fopen( IO_TRACE_DUMP_FILE_NAME, "wt" );
13:    else out = fopen( dataset->trace.filename, "wt" );
14:
15:    error_code = -EC_FILE_ERROR;
16:    if ( NULL == out ) return error_code;
17:    else error_code = SUCCESS;
18:
19:    index = dataset->trace.index;
20:
21:
22:
23:    count = dataset->trace.count;
24:
25:
26:
27:
28:
29:    {
30:        char buf[64]={ 0 };
31:        Time_Current_String( buf, 64 );
32:        fprintf( out, "%s\n", buf );
33:    }
34:    fprintf( out, "trace.start      = " );
35:    fprintf( out, "%s\n", global_io_trace_start_name[dataset->trace.start] );
36:    fprintf( out, "trace.stop       = " );
37:    fprintf( out, "%s\n", global_io_trace_stop_name[dataset->trace.stop] );
38:    fprintf( out, "trace.file_name = " );
39:    if ( 0 == strlen( dataset->trace.filename ) ) fprintf( out, "%s\n",
IO_TRACE_DUMP_FILE_NAME );
40:    else fprintf( out, "%s\n",
dataset->trace.filename );
41:
42:
43:
44:    fprintf( out, "\row\" );
45:    fprintf( out, ", \"posting\" );
46:    fprintf( out, ", \"board\" );
47:    fprintf( out, ", \"action\" );
48:    fprintf( out, ", \"src_line\" );

```

```
49:     fprintf( out, ", \"register_or_function\" );
50:     fprintf( out, ", \"channel\" );
51:     fprintf( out, ", \"error\" );
52:     fprintf( out, ", \"value\" );
53:     fprintf( out, "\n" );
54:
55:     row = 0;
56:     while ( count > 0 )
57:     {
58:         trace = &(dataset->trace.buf[index]);
59:
60:
61:         fprintf( out, "%3u", (unsigned) row );
62:
63:
64:         fprintf( out, " %6u", (unsigned) trace->posting );
65:
66:
67:
68:         fprintf( out, " " );
69:         switch( trace->board_id )
70:         {
71:             case ID_IDI48: fprintf( out, "\"IDI48\"" ); break;
72:             case ID_IDO48: fprintf( out, "\"IDO48\"" ); break;
73:             case ID_IAI16: fprintf( out, "\"IAI16\"" ); break;
74:             case ID_ARM32: fprintf( out, "\"ARM32\"" ); break;
75:         }
76:
77:         fprintf( out, " " );
78:         fprintf( out, "\"%s\"", global_io_trace_name[trace->action] );
79:
80:
81:         fprintf( out, " %6u", (unsigned) trace->line );
82:
83:
84:         switch( trace->action )
85:         {
86:             case IO_TRACE_READ:
87:             case IO_TRACE_WRITE:
88:                 fprintf( out, ", \"reg: %s\"", definition[REG_LOCATION_LOGICAL_GET(trace->location) +
REG_LOCATION_CHANNEL_GET( trace->location )].symbol_name );
89:                 break;
90:             default:
91:                 if ( NULL != trace->function_name )
92:                 {
93:                     fprintf( out, ", \"fnc: %s\"", trace->function_name );
94:                 }
95:                 break;
96:             }
97:
98:
99:         fprintf( out, ", %d", REG_LOCATION_CHANNEL_GET( trace->location ) );
100:
101:        fprintf( out, ", \"%s\"", EC_Code_To_Human_Readable( trace->error_code ) );
102:
103:        fprintf( out, ", 0x%02X", (int) trace->value );
104:
105:        fprintf( out, "\n" );
106:
107:
108:        if ( index == dataset->trace.begin )
109:        {
110:            index = dataset->trace.end;
111:        }
```

```

112: else
113: {
114:     index--;
115: }
116: count--;
117: row++;
118: }
119:
120: fclose( out );
121: return error_code;
122: }
```

3.2.1.191 IO_Trace_Initialize Function

C++

```
int IO_Trace_Initialize(int board_id);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: int IO_Trace_Initialize( int board_id )
2: {
3:     size_t      index;
4:     struct io_trace *   trace;
5:     struct board_dataset * dataset;
6:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
7:
8:     dataset->trace.begin = 0;
9:     dataset->trace.end   = IO_TRACE_SIZE - 1;
10:    dataset->trace.index = 0;
11:    dataset->trace.count = 0;
12:    dataset->trace.wrap  = false;
13:    dataset->trace.posting = 0;
14:
15:    if ( IO_TRACE_START_ON == dataset->trace.start ) dataset->trace.active = true;
16:    else                                         dataset->trace.active = false;
17:
18:    dataset->trace.buf = &(global_io_trace_buf[0]);
19:
20:    for ( index = 0; index < IO_TRACE_SIZE; index++ )
21:    {
22:        trace = &(dataset->trace.buf[index]);
23:        memset( trace, 0, sizeof( struct io_trace ) );
24:        dataset->trace.buf[index].action   = IO_TRACE_NULL;
25:        dataset->trace.buf[index].function_name = NULL;
26:        dataset->trace.buf[index].board_id   = ID_BOARD_UNKNOWN;
27:    }
28:
29:    if ( strlen(dataset->trace.filename) == 0 )
30:    {
31:        strcpy( dataset->trace.filename, IO_TRACE_DUMP_FILE_NAME );
32:    }
33:    return SUCCESS;
```

```
34: }
```

3.2.1.192 IO_Trace_Log Function

C++

```
int IO_Trace_Log(int board_id, size_t line, enum IO_TRACE_ENUM action, int location, int
error_code, uint8_t value);
```

File

idi.c (see page 373)

Description

brief

Body Source

```
1: int IO_Trace_Log( int board_id, size_t line, enum IO_TRACE_ENUM action, int location, int
error_code, uint8_t value )
2: {
3:     int ec;
4:     if ( true == global_io_trace_enable )
5:     {
6:         struct board_dataset * dataset;
7:         dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
8:
9:         ec = IO_Trace_Log_Dataset( dataset, line, action, location, error_code, value );
10:    }
11:    else
12:    {
13:        ec = SUCCESS;
14:    }
15:    return ec;
16: }
```

3.2.1.193 IO_Trace_Log_Dataset Function

C++

```
int IO_Trace_Log_Dataset(struct board_dataset * ds, size_t line, enum IO_TRACE_ENUM action,
int location, int error_code, uint8_t value);
```

File

idi.c (see page 373)

Description

brief

Body Source

```
1: int IO_Trace_Log_Dataset( struct board_dataset * ds, size_t line, enum IO_TRACE_ENUM
action, int location, int error_code, uint8_t value )
2: {
```

```

3: struct io_trace * trace;
4:
5: if ( 0 != IO_Trace_Log_Dataset_Begin( ds, line, action, location, error_code, value ) )
return SUCCESS;
6:
7:
8:
9:
10:
11:
12:
13:
14:
15:
16:
17:
18:
19:
20:
21:
22:
23:
24:
25:
26:
27:
28:
29:
30:
31: trace = &(ds->trace.buf[ds->trace.index]);
32: trace->action    = action;
33: trace->board_id  = ds->board_id;
34: trace->location   = location;
35: trace->error_code = error_code;
36: trace->value     = value;
37: trace->posting   = ds->trace.posting;
38: trace->line     = line;
39: ds->trace.posting++;
40:
41: IO_Trace_Log_Dataset_End( ds, line, action, location, error_code, value );
42:
43:
44:
45:
46:
47:
48:
49: return SUCCESS;
50: }
```

3.2.1.194 IO_Trace_Log_Dataset_Begin Function

C++

```
static int IO_Trace_Log_Dataset_Begin(struct board_dataset * ds, size_t line, enum
IO_TRACE_ENUM action, int location, int error_code, uint8_t value);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int IO_Trace_Log_Dataset_Begin( struct board_dataset * ds, size_t line, enum
IO_TRACE_ENUM action, int location, int error_code, uint8_t value )
2: {
3:   (void) line;
4:   (void) location;
5:   (void) value;
6:
7:   if ( false == global_io_trace_enable ) return 1;
8:
9:
10:  switch( ds->trace.start )
11:  {
12:    IO_TRACE_START_DEFINITION( IO_TRACE_START_STOP_EXTRACT_SWITCH_STATEMENT )
13:  }
14:
15:  if ( false == ds->trace.active ) return 2;
16:
17:
18:  if ( false == ds->trace.wrap ) ds->trace.count++;
19:
20:  if ( ds->trace.index >= ds->trace.end )
21:  {
22:    ds->trace.index = ds->trace.begin;
23:    ds->trace.wrap = true;
24:  }
25:  else
26:  {
27:    ds->trace.index++;
28:  }
29:  return 0;
30: }
```

3.2.1.195 IO_Trace_Log_Dataset_End Function

C++

```
static int IO_Trace_Log_Dataset_End(struct board_dataset * ds, size_t line, enum IO_TRACE_ENUM
action, int location, int error_code, uint8_t value);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int IO_Trace_Log_Dataset_End( struct board_dataset * ds, size_t line, enum
IO_TRACE_ENUM action, int location, int error_code, uint8_t value )
2: {
3:   (void) line;
```

```

4:  (void) action;
5:  (void) location;
6:  (void) value;
7:
8:
9:  switch( ds->trace.stop )
10: {
11:   IO_TRACE_STOP_DEFINITION( IO_TRACE_START_STOP_EXTRACT_SWITCH_STATEMENT )
12: }
13: return SUCCESS;
14: }
```

3.2.1.196 IO_Trace_Log_Function Function

C++

```
int IO_Trace_Log_Function(const char * function_name, size_t line, enum IO_TRACE_ENUM action,
int board_id, int error_code);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: int IO_Trace_Log_Function( const char * function_name, size_t line, enum IO_TRACE_ENUM
action, int board_id, int error_code )
2: {
3:   int ec;
4:   if ( true == global_io_trace_enable )
5:   {
6:     struct io_trace * trace;
7:     struct board_dataset * ds;
8:     ds = ( struct board_dataset * ) board_definition[board_id].dataset;
9:
10:    if ( 0 != IO_Trace_Log_Dataset_Begin( ds, line, action, 0, error_code, 0 ) ) return
SUCCESS;
11:
12:    trace = &(ds->trace.buf[ds->trace.index]);
13:    trace->action      = action;
14:    trace->board_id    = ds->board_id;
15:    trace->location    = 0;
16:    trace->error_code  = error_code;
17:    trace->value       = 0;
18:    trace->posting     = ds->trace.posting;
19:    trace->line       = line;
20:    trace->function_name = function_name;
21:    ds->trace.posting++;
22:
23:    IO_Trace_Log_Dataset_End( ds, line, action, 0, error_code, 0 );
24:  }
25: else
26:  {
27:    ec = SUCCESS;
28:  }
29: return ec;
30: }
```

3.2.1.197 IO_Write_U16_Address_Fixed Function

C++

```
void IO_Write_U16_Address_Fixed(int board_id, size_t line, int location, uint16_t value);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Writes uint16_t (see page 321) to I/O ports in a uint8_t (see page 321) succession to the same address location. Macros are used to guide the target implementation. In this case, bus width (which we typically refer to the port width, which is different than register width) is assumed to be byte (uint8_t (see page 321)) wide.

param[in] location the enumerated register symbol. The enumerated symbol is composed of offset and bank information used to determine the final address information. param[in] value The uint16_t (see page 321) value to be written to the I/O register

Body Source

```
1: void IO_Write_U16_Address_Fixed( int board_id, size_t line, int location, uint16_t value )
2: {
3:
4:   IO_Write_U8( board_id, line, (int)((int) location) + 0 ), (uint8_t)(      value      & 0xFF
) );
5:   IO_Write_U8( board_id, line, (int)((int) location) + 0 ), (uint8_t)( ( value >> 8 ) & 0xFF
) );
6: }
```

3.2.1.198 IO_Write_U16_Address_Increment Function

C++

```
void IO_Write_U16_Address_Increment(int board_id, size_t line, int location, uint16_t value);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Writes uint16_t (see page 321) to I/O ports in a uint8_t (see page 321) succession incrementing the offset address. Macros are used to guide the target implementation. In this case, bus width (which we typically refer to the port width, which is different than register width) is assumed to be byte (uint8_t (see page 321)) wide.

param[in] location the enumerated register symbol. The enumerated symbol is composed of offset and bank information used to determine the final address information. param[in] value The uint16_t (see page 321) value to be written to the I/O register

Body Source

```

1: void IO_Write_U16_Address_Increment( int board_id, size_t line, int location, uint16_t
value )
2: {
3:
4:
5:
6: IO_Write_U8( board_id, line, (int)((int) location) + 0 ), (uint8_t)( value &
0xFF );
7: IO_Write_U8( board_id, line, (int)((int) location) + 1 ), (uint8_t)( ( value >> 8 ) &
0xFF );
8: }
```

3.2.1.199 IO_Write_U8 Function

C++

```
int IO_Write_U8(int board_id, size_t line, int location, uint8_t value);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Writes uint8_t (see page 321) to I/O port. Macros are used to guide the target implementation.

param[in] board_id software level board selector or descriptor. param[in] location the enumerated register symbol. The enumerated symbol is composed of offset and bank information used to determine the final address information. param[in] value The data to be written out.

Body Source

```

1: int IO_Write_U8( int board_id, size_t line, int location, uint8_t value )
2: {
3: int error_code;
4: int index;
5: int offset;
6: int channel;
7: int address;
8: int bank;
9:
10: struct reg_definition * definition = ( struct reg_definition * )
board_definition[board_id].definition;
11: struct board_dataset * dataset = ( struct board_dataset * )
board_definition[board_id].dataset;
12:
13: error_code = SUCCESS;
14: index = REG_LOCATION_LOGICAL_GET( location );
15: #if defined( IDI_IO_DIRECTION_TEST )
16: if ( !( REG_DIR_WRITE == ( definition[index].direction & REG_DIR_WRITE ) ) )
```

```

17: {
18:     printf( "IO_Write_U8: %s, error in direction\n", definition[index].symbol_name );
19:     error_code = -EC_DIRECTION;
20:     goto IO_Write_U8_Exit;
21: }
22: #endif
23:     bank    = definition[index].bank;
24:     if ( ( BANK_NONE != bank ) && ( bank != dataset->bank_previous ) )
25:     {
26:         address = dataset->base_address + definition[REG_LOCATION_LOGICAL_GET(
board_definition[board_id].bank_register_symbol)].physical_offset;
27:         dataset->bank_previous = bank;
28:         IO_Write_U8_Port( dataset, __LINE__, address, (uint8_t) bank );
29:         IO_Trace_Log( board_id, line, IO_TRACE_WRITE,
board_definition[board_id].bank_register_symbol, error_code, (uint8_t) bank );
30:         if ( ( dataset->io_report ) || ( dataset->io_simulate ) )
31:         {
32:             printf( "IO_Write_U8 bank write: %s, address = 0x%04X, bank = %s (0x%02X)\n",
definition[index].symbol_name,
33:                     address,
34:                     Bank_Symbol_To_Name( board_id, bank ),
35:                     bank
36:                 );
37:         }
38:     }
39:     channel = REG_LOCATION_CHANNEL_GET( location );
40:     offset   = definition[index].physical_offset;
41:     address  = dataset->base_address + offset + channel;
42:     IO_Write_U8_Port( dataset, __LINE__, address, value );
43:     IO_Write_U8_Exit:
44:     IO_Trace_Log( board_id, line, IO_TRACE_WRITE, location, error_code, value );
45:     return SUCCESS;
46: }
```

3.2.1.200 IO_Write_U8_Port Function

C++

```
static int IO_Write_U8_Port(struct board_dataset * dataset, size_t line, int address, uint8_t value);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int IO_Write_U8_Port( struct board_dataset * dataset, size_t line, int address,
uint8_t value )
2: {
3:     if ( NULL == dataset )
4:     {
5: #if defined( __MSDOS__ )
6:         outportb( address, value );
7: #else
8:         printf( "IO_Write_U8_Port: address = 0x%04X, value = 0x%02X\n", address, value );
9: #endif

```

```

10: }
11: else
12: {
13:     if ( !dataset->io_simulate )
14:     {
15: #if defined( __MSDOS__ )
16:         outportb( address, value );
17: #endif
18:     }
19:     if ( ( dataset->io_report ) || ( dataset->io_simulate ) )
20:     {
21:         printf( "IO_Write_U8_Port: address = 0x%04X, value = 0x%02X\n", address, value );
22:     }
23: }
24: return SUCCESS;
25: }
```

3.2.1.201 IOKern_DOS_IRQ_Free Function

C++

```
void IOKern_DOS_IRQ_Free(unsigned int irq);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: void IOKern_DOS_IRQ_Free( unsigned int irq
2:
3:     )
4: {
5:     IOKERN_TASK_TYPE * task;
6:
7: #define IOKERN_FUNCTION_NAME "IOKern_DOS_IRQ_Free"
8: #if defined ( MSDOS_PRINTF_DEBUG_MESSAGES )
9:     Printf( "%s: ", IOKERN_FUNCTION_NAME );
10:    Printf( "irq=%d, ", irq );
11: #endif
12:    task = IOKern_Task_Handle_Get( irq );
13:    if ( ( SUCCESS == IOKern_IRQ_Test( irq ) ) && ( NULL != task ) )
14:    {
15:        if ( task->task_fp )
16:        {
17:            IOKern_DOS_ISR_Restore( (unsigned char) irq );
18:            memset( task, 0, sizeof( IOKERN_TASK_TYPE ) );
19:            task->number = IOKERN_IRQ_NONE;
20: #if defined ( MSDOS_PRINTF_DEBUG_MESSAGES )
21:            Printf( "success " );
22: #endif
23:    }
```

```

24: }
25: else
26: {
27: #if defined ( MSDOS_PRINTF_DEBUG_MESSAGES )
28: Printf( " " );
29: #endif
30: }
31: #if defined ( MSDOS_PRINTF_DEBUG_MESSAGES )
32: Printf( "done\n" );
33: #endif
34: #undef IOKERN_FUNCTION_NAME
35: }
```

3.2.1.202 IOKern_DOS_IRQ_Mask_Get Function

C++

```
static void IOKern_DOS_IRQ_Mask_Get(unsigned char irq, unsigned char * state);
```

File

idi.c (see page 373)

Body Source

```

1: static void IOKern_DOS_IRQ_Mask_Get( unsigned char irq, unsigned char * state )
2: {
3:     unsigned int port;
4:     unsigned char value;
5:     if ( irq < 8 )
6:     {
7:         port = IOKERN_DOS_PIC1_IMR;
8:     }
9:     else
10:    {
11:        irq = irq - 8;
12:        port = IOKERN_DOS_PIC2_IMR;
13:    }
14:    value = inp( port ) & ( 1 << irq );
15:    *state = value;
16: }
```

3.2.1.203 IOKern_DOS_IRQ_Mask_Set Function

C++

```
static void IOKern_DOS_IRQ_Mask_Set(unsigned char irq, unsigned char state);
```

File

idi.c (see page 373)

Parameters

Parameters	Description
unsigned char state	0=On,1=Off

Body Source

```

1: static void IOKern_DOS IRQ_Mask_Set( unsigned char irq, unsigned char state )
2: {
3:     unsigned int port;
4:     unsigned char value;
5:
6:     port = IOKERN_DOS_PIC1_IMR;
7:     if ( irq >= 8 )
8:     {
9:
10:        if ( state ) value = inp( port ) | ( 1 << 2 );
11:        else          value = inp( port ) & ~( 1 << 2 );
12:        outp( port, value );
13:
14:        irq = irq - 8;
15:        port = IOKERN_DOS_PIC2_IMR;
16:    }
17:    if ( state ) value = inp( port ) | ( 1 << irq );
18:    else          value = inp( port ) & ~( 1 << irq );
19:    outp( port, value );
20: }
```

3.2.1.204 IOKern_DOS_IRQ_Request Function

C++

```
int IOKern_DOS_IRQ_Request(unsigned int irq, IOKERN_TASK_FP handler, void * dev_id);
```

File

idi.c (see page 373)

Parameters

Parameters	Description
IOKERN_TASK_FP handler	unsigned long flags, const char * dev_name,
void * dev_id	private data

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: int IOKern_DOS IRQ_Request( unsigned int irq,
2:                             IOKERN_TASK_FP handler,
3:
4:
5:                             void *     dev_id
6:                             )
7: {
8:     int      error_code;
```

```
9: IOKERN_TASK_ID_ENUM    task_id;
10: #define IOKERN_FUNCTION_NAME "IOKern_DOS_IRQ_Request"
11: #if defined ( MSDOS_PRINTF_DEBUG_MESSAGES )
12: Printf( "%s: ", IOKERN_FUNCTION_NAME );
13: Printf( "irq=%d, ", irq );
14: #endif
15: #undef IOKERN_FUNCTION_NAME
16: if ( SUCCESS != IOKern_IRQ_Test( irq ) ) return -EC_INTR_ERROR;
17:
18: task_id = IOKern_Task_ID_Get( irq );
19: if ( IOKERN_TASK_ID_NONE == task_id ) return SUCCESS;
20:
21: if ( NULL == handler ) return -EC_INTR_ERROR;
22: if ( NULL != iokern_task[task_id].task_fp ) return -EC_BUSY;
23:
24: iokern_task[task_id].task_fp = handler;
25:
26:
27:
28:
29:
30:
31:
32:
33:
34:
35:
36:
37:
38:
39:
40:
41:
42:
43:
44: iokern_task[task_id].help_fp  = NULL;
45: iokern_task[task_id].chain_to_old = IOKERN_CHAIN_TO_OLD_OFF;
46:
47:
48: iokern_task[task_id].dev_id  = dev_id;
49:
50: iokern_task[task_id].number  = irq;
51: error_code = IOKern_DOS_ISR_Install( (unsigned char) irq );
52: #if defined ( MSDOS_PRINTF_DEBUG_MESSAGES )
53: Printf( "done\n" );
54: #endif
55: return error_code;
56: #undef IOKERN_FUNCTION_NAME
57: }
```

3.2.1.205 IOKern_DOS_ISR_Install Function

C++

```
static int IOKern_DOS_ISR_Install(unsigned char irq);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int IOKern_DOS_ISR_Install( unsigned char irq )
2: {
3:     IOKERN_TASK_ID_ENUM task_id;
4:     int int_number;
5:
6:     task_id = IOKern_Task_ID_Get( irq );
7:     if ( IOKERN_TASK_ID_NONE == task_id ) return SUCCESS;
8:
9:     if ( ( IOKERN_TASK_ID_IRQ0 == task_id ) || ( IOKERN_TASK_ID_INT33 == task_id ) )
10:    {
11:        return -EC_INTERRUPT_UNAVAILABLE;
12:    }
13:
14:
15:
16:
17:
18:
19:
20:
21:
22:
23:
24:
25:
26:
27:
28:     if ( irq < 8 ) int_number = ( irq - 0 ) + 0x08;
29:     else           int_number = ( irq - 8 ) + 0x70;
30:
31:
32:     iokern_task[task_id].old_isr = (IOKERN_ISR_FP) IOKERN_DOS_INT_GET( int_number );
33:     IOKern_DOS IRQ_Mask_Get( irq, &(iokern_task[task_id].old_state) );
34:     IOKERN_DOS INT_Set( int_number, iokern_isr_table[task_id] );
35:     IOKern_DOS IRQ_Mask_Set( irq, 0 );
36:
37:
38:     return SUCCESS;
39: }
```

3.2.1.206 IOKern_DOS_ISR_Restore Function

C++

```
static void IOKern_DOS_ISR_Restore(unsigned char irq);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static void IOKern_DOS_ISR_Restore( unsigned char irq )
2: {
3:     IOKERN_TASK_ID_ENUM task_id;
4:     int int_number;
5:
6:     task_id = IOKern_Task_ID_Get( irq );
7:     if ( IOKERN_TASK_ID_NONE == task_id ) return;
8:
9:
10:
11:
12:
13:
14:
15:
16:
17:     if ( irq < 8 ) int_number = ( irq - 0 ) + 0x08;
18:     else           int_number = ( irq - 8 ) + 0x70;
19:
20:     IOKern_DOS_IRQ_Mask_Set( irq, iokern_task[task_id].old_state );
21:     IOKERN_DOS_INT_SET( int_number, iokern_task[task_id].old_isr );
22:
23: }
```

3.2.1.207 IOKern_DOS_Vector_Get Function

C++

```
IOKERN_DOS_VECTOR_TYPE IOKern_DOS_Vector_Get(unsigned num);
```

File

idi.c (see page 373)

Body Source

```

1: IOKERN_DOS_VECTOR_TYPE IOKern_DOS_Vector_Get( unsigned num )
2: {
3:     asm push es
4:     asm push bx
5:
6:     asm xor bx,bx
7:     asm mov es,bx
8:     asm mov bx,[num]
9:
10:    asm shl bx,1
11:    asm shl bx,1
12:
13:    asm les bx,es:[bx]
```

```

16:
17:     asm mov dx,es
18:     asm mov ax,bx
19:     asm pop bx
20:     asm pop es
21:
22: }
```

3.2.1.208 IOKern_DOS_Vector_Set Function

C++

```
void IOKern_DOS_Vector_Set(unsigned num, IOKERN_DOS_VECTOR_TYPE h);
```

File

idi.c (see page 373)

Body Source

```

1: void IOKern_DOS_Vector_Set( unsigned num, IOKERN_DOS_VECTOR_TYPE h )
2: {
3:     asm push es
4:     asm push bx
5:
6:     asm xor bx,bx
7:     asm mov es,bx
8:     asm mov bx,[num]
9:
10:    asm shl bx,1
11:    asm shl bx,1
12:
14:    asm pushf
15:    asm cli
16:    asm mov ax,word ptr [h]
17:    asm mov es:[bx],ax
18:    asm mov ax,word ptr [h+2]
19:    asm mov es:[bx+2],ax
20:    asm popf
21:    asm pop bx
22:    asm pop es
23: }
```

3.2.1.209 IOKern_Interrupt_Helper Function

C++

```
void IOKern_Interrupt_Helper(IOKERN_TASK_ID_ENUM id, IOKERN_IRQ_ENUM irq, struct pt_regs * regs);
```

File

idi.c (see page 373)

Description

brief Supporting 'fast' interrupts

Body Source

```
1: void IOKern_Interrupt_Helper( IOKERN_TASK_ID_ENUM id, IOKERN_IRQ_ENUM irq, struct pt_regs *  
regs )  
2: {  
3:   iokern_task[id].result =  
4:     ( * iokern_task[id].task_fp )( irq, iokern_task[id].dev_id, regs );  
5:  
6:   iokern_task[id].count++;  
7: }
```

3.2.1.210 IOKern_IRQ_Invoke_ISR_Test Function

C++

```
void IOKern_IRQ_Invoke_ISR_Test(size_t irq);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```
1: void IOKern_IRQ_Invoke_ISR_Test( size_t irq )  
2: {  
3:   struct pt_regs r;  
4:  
5:   memset( &r, 0, sizeof( struct pt_regs ) );  
6:   ( * iokern_isr_table[irq] )( r );  
7: }
```

3.2.1.211 IOKern_IRQ_Test Function

C++

```
int IOKern_IRQ_Test(unsigned int irq);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: int IOKern_IRQ_Test( unsigned int irq )
2: {
3:     int error_code;
4:
5:     error_code = IOKern_IRQ_Test_Helper( irq );
6:
7:     if ( error_code ) return -EC_INTR_ERROR;
8:
9:     return SUCCESS;
10: }
11: }
```

3.2.1.212 IOKern_IRQ_Test_Helper Function

C++

```
int IOKern_IRQ_Test_Helper(unsigned int irq);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: int IOKern_IRQ_Test_Helper( unsigned int irq )
2: {
3:     int error_code = SUCCESS;
4: #if defined( IOKERN_CPU_REGION_MAP )
5:     int index;
6:
7:     index = 0;
8:     while( SYS_TYPE__NONE != iokern_irq_map_list[index].type )
9:     {
10:         if ( irq == iokern_irq_map_list[index].irq )
11:         {
12:             return( -((int)(index + 1)) );
13:         }
14:         index++;
15:     }
16: #endif
17:
18: switch( (IOKERN_IRQ_ENUM) irq )
19: {
20:     case IOKERN_IRQ_NONE:
21: }
```

```

22:   case IOKERN_IRQ_1:
23:
24:   case IOKERN_IRQ_3:
25:   case IOKERN_IRQ_4:
26:   case IOKERN_IRQ_5:
27:   case IOKERN_IRQ_6:
28:   case IOKERN_IRQ_7:
29:   case IOKERN_IRQ_9:
30:   case IOKERN_IRQ_10:
31:   case IOKERN_IRQ_11:
32:   case IOKERN_IRQ_12:
33:
34:   case IOKERN_IRQ_14:
35:   case IOKERN_IRQ_15:
36:
37:     break;
38:   default:
39:     error_code = -EC_INTR_ERROR;
40: }
41:
42: return error_code;
43: }
```

3.2.1.213 IOKern_ISR0 Function

C++

```
static void INTERRUPT IOKern_ISR0(struct pt_regs r);
```

File

idi.c (see page 373)

Description

brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.

Body Source

```

1: static void INTERRUPT IOKern_ISR0( struct pt_regs r )
2: {
3: #if defined( IOKERN_ISR_USE_THESE_GUTS )
4:   IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ1, IOKERN_IRQ_0, r );
5: #else
6:   IOKERN_IRQ_START;
7:   IOKern Interrupt_Helper( IOKERN_TASK_ID_IRQ0, IOKERN_IRQ_0, &r );
8:   IOKERN_IRQ_END( IOKERN_IRQ_0 );
9: #endif
10: }
```

3.2.1.214 IOKern_ISR1 Function

C++

```
static void INTERRUPT IOKern_ISR1(struct pt_regs r);
```

File

idi.c (see page 373)

Description

brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.

Body Source

```

1: static void INTERRUPT IOKern_ISR1( struct pt_regs r )
2: {
3: #if defined( IOKERN_ISR_USE_THESE_GUTS )
4: IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ1, IOKERN_IRQ_1, r );
5: #else
6: IOKERN_IRQ_START;
7: IOKern_Interrupt_Helper( IOKERN_TASK_ID_IRQ1, IOKERN_IRQ_1, &r );
8: IOKERN_IRQ_END( IOKERN_IRQ_1 );
9: #endif
10: }
```

3.2.1.215 IOKern_ISR10 Function

C++

```
static void INTERRUPT IOKern_ISR10(struct pt_regs r);
```

File

idi.c (see page 373)

Description

brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.

Body Source

```

1: static void INTERRUPT IOKern_ISR10( struct pt_regs r )
2: {
3: #if defined( IOKERN_ISR_USE_THESE_GUTS )
4: IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ10, IOKERN_IRQ_10, r );
5: #else
6: IOKERN_IRQ_START;
7: IOKern_Interrupt_Helper( IOKERN_TASK_ID_IRQ10, IOKERN_IRQ_10, &r );
8: IOKERN_IRQ_END( IOKERN_IRQ_10 );
9: #endif
10: }
```

3.2.1.216 IOKern_ISR11 Function

C++

```
static void INTERRUPT IOKern_ISR11(struct pt_regs r);
```

File

idi.c (see page 373)

Description

brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.

Body Source

```

1: static void INTERRUPT IOKern_ISR11( struct pt_regs r )
2: {
3: #if defined( IOKERN_ISR_USE_THESE_GUTS )
4: IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ11, IOKERN_IRQ_11, r );
5: #else
6: IOKERN_IRQ_START;
7: IOKern_Interrupt_Helper( IOKERN_TASK_ID_IRQ11, IOKERN_IRQ_11, &r );
8: IOKERN_IRQ_END( IOKERN_IRQ_11 );
9: #endif
10: }
```

3.2.1.217 IOKern_ISR12 Function

C++

```
static void INTERRUPT IOKern_ISR12(struct pt_regs r);
```

File

idi.c (see page 373)

Description

brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.

Body Source

```

1: static void INTERRUPT IOKern_ISR12( struct pt_regs r )
2: {
3: #if defined( IOKERN_ISR_USE_THESE_GUTS )
4: IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ12, IOKERN_IRQ_12, r );
5: #else
6: IOKERN_IRQ_START;
7: IOKern_Interrupt_Helper( IOKERN_TASK_ID_IRQ12, IOKERN_IRQ_12, &r );
8: IOKERN_IRQ_END( IOKERN_IRQ_12 );
9: #endif
10: }
```

3.2.1.218 IOKern_ISR13 Function

C++

```
static void INTERRUPT IOKern_ISR13(struct pt_regs r);
```

File

idi.c (see page 373)

Description

brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.

Body Source

```

1: static void INTERRUPT IOKern_ISR13( struct pt_regs r )
2: {
3: #if defined( IOKERN_ISR_USE_THESE_GUTS )
4: IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ13, IOKERN_IRQ_13, r );
5: #else
6: IOKERN_IRQ_START;
7: IOKern_Interrupt_Helper( IOKERN_TASK_ID_IRQ13, IOKERN_IRQ_13, &r );
8: IOKERN_IRQ_END( IOKERN_IRQ_13 );
9: #endif
10: }
```

3.2.1.219 IOKern_ISR14 Function

C++

```
static void INTERRUPT IOKern_ISR14(struct pt_regs r);
```

File

idi.c (see page 373)

Description

brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.

Body Source

```

1: static void INTERRUPT IOKern_ISR14( struct pt_regs r )
2: {
3: #if defined( IOKERN_ISR_USE_THESE_GUTS )
4: IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ14, IOKERN_IRQ_14, r );
5: #else
6: IOKERN_IRQ_START;
7: IOKern_Interrupt_Helper( IOKERN_TASK_ID_IRQ14, IOKERN_IRQ_14, &r );
8: IOKERN_IRQ_END( IOKERN_IRQ_14 );
9: #endif
10: }
```

3.2.1.220 IOKern_ISR15 Function

C++

```
static void INTERRUPT IOKern_ISR15(struct pt_regs r);
```

File

idi.c (see page 373)

Description

brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.

Body Source

```

1: static void INTERRUPT IOKern_ISR15( struct pt_regs r )
2: {
3: #if defined( IOKERN_ISR_USE_THESE_GUTS )
4: IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ15, IOKERN_IRQ_15, r );
5: #else
6: IOKERN_IRQ_START;
7: IOKern_Interrupt_Helper( IOKERN_TASK_ID_IRQ15, IOKERN_IRQ_15, &r );
8: IOKERN_IRQ_END( IOKERN_IRQ_15 );
9: #endif
10: }
```

3.2.1.221 IOKern_ISR2 Function

C++

```
static void INTERRUPT IOKern_ISR2(struct pt_regs r);
```

File

idi.c (see page 373)

Description

brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.

Body Source

```

1: static void INTERRUPT IOKern_ISR2( struct pt_regs r )
2: {
3: #if defined( IOKERN_ISR_USE_THESE_GUTS )
4: IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ2, IOKERN_IRQ_2, r );
5: #else
6: IOKERN_IRQ_START;
7: IOKern_Interrupt_Helper( IOKERN_TASK_ID_IRQ2, IOKERN_IRQ_2, &r );
8: IOKERN_IRQ_END( IOKERN_IRQ_2 );
9: #endif
10: }
```

3.2.1.222 IOKern_ISR3 Function

C++

```
static void INTERRUPT IOKern_ISR3(struct pt_regs r);
```

File

idi.c (see page 373)

Description

brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.

Body Source

```

1: static void INTERRUPT IOKern_ISR3( struct pt_regs r )
2: {
3: #if defined( IOKERN_ISR_USE_THESE_GUTS )
4: IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ3, IOKERN_IRQ_3, r );
5: #else
6: IOKERN_IRQ_START;
7: IOKern_Interrupt_Helper( IOKERN_TASK_ID_IRQ3, IOKERN_IRQ_3, &r );
8: IOKERN_IRQ_END( IOKERN_IRQ_3 );
9: #endif
10: }
```

3.2.1.223 IOKern_ISR4 Function

C++

```
static void INTERRUPT IOKern_ISR4(struct pt_regs r);
```

File

idi.c (see page 373)

Description

brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.

Body Source

```

1: static void INTERRUPT IOKern_ISR4( struct pt_regs r )
2: {
3: #if defined( IOKERN_ISR_USE_THESE_GUTS )
4: IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ4, IOKERN_IRQ_4, r );
5: #else
6: IOKERN_IRQ_START;
7: IOKern_Interrupt_Helper( IOKERN_TASK_ID_IRQ4, IOKERN_IRQ_4, &r );
8: IOKERN_IRQ_END( IOKERN_IRQ_4 );
9: #endif
10: }
```

3.2.1.224 IOKern_ISR5 Function

C++

```
static void INTERRUPT IOKern_ISR5(struct pt_regs r);
```

File

idi.c (see page 373)

Description

brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.

Body Source

```
1: static void INTERRUPT IOKern_ISR5( struct pt_regs r )
2: {
3: #if defined( IOKERN_ISR_USE_THESE_GUTS )
4: IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ5, IOKERN_IRQ_5, r );
5: #else
6: IOKERN_IRQ_START;
7: IOKern_Interrupt_Helper( IOKERN_TASK_ID_IRQ5, IOKERN_IRQ_5, &r );
8: IOKERN_IRQ_END( IOKERN_IRQ_5 );
9: #endif
10: }
```

3.2.1.225 IOKern_ISR6 Function

C++

```
static void INTERRUPT IOKern_ISR6(struct pt_regs r);
```

File

idi.c (see page 373)

Description

brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.

Body Source

```
1: static void INTERRUPT IOKern_ISR6( struct pt_regs r )
2: {
3: #if defined( IOKERN_ISR_USE_THESE_GUTS )
4: IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ6, IOKERN_IRQ_6, r );
5: #else
6: IOKERN_IRQ_START;
7: IOKern_Interrupt_Helper( IOKERN_TASK_ID_IRQ6, IOKERN_IRQ_6, &r );
8: IOKERN_IRQ_END( IOKERN_IRQ_6 );
9: #endif
10: }
```

3.2.1.226 IOKern_ISR7 Function

C++

```
static void INTERRUPT IOKern_ISR7(struct pt_regs r);
```

File

idi.c (see page 373)

Description

brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.

Body Source

```

1: static void INTERRUPT IOKern_ISR7( struct pt_regs r )
2: {
3: #if defined( IOKERN_ISR_USE_THESE_GUTS )
4: IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ7, IOKERN_IRQ_7, r );
5: #else
6: IOKERN_IRQ_START;
7: IOKern_Interrupt_Helper( IOKERN_TASK_ID_IRQ7, IOKERN_IRQ_7, &r );
8: IOKERN_IRQ_END( IOKERN_IRQ_7 );
9: #endif
10: }
```

3.2.1.227 IOKern_ISR8 Function

C++

```
static void INTERRUPT IOKern_ISR8(struct pt_regs r);
```

File

idi.c (see page 373)

Description

brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.

Body Source

```

1: static void INTERRUPT IOKern_ISR8( struct pt_regs r )
2: {
3: #if defined( IOKERN_ISR_USE_THESE_GUTS )
4: IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ8, IOKERN_IRQ_8, r );
5: #else
6: IOKERN_IRQ_START;
7: IOKern_Interrupt_Helper( IOKERN_TASK_ID_IRQ8, IOKERN_IRQ_8, &r );
8: IOKERN_IRQ_END( IOKERN_IRQ_8 );
9: #endif
10: }
```

3.2.1.228 IOKern_ISR9 Function

C++

```
static void INTERRUPT IOKern_ISR9(struct pt_regs r);
```

File

idi.c (see page 373)

Description

brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.

Body Source

```
1: static void INTERRUPT IOKern_ISR9( struct pt_regs r )
2: {
3: #if defined( IOKERN_ISR_USE THESE_GUTS )
4: IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ9, IOKERN_IRQ_9, r );
5: #else
6: IOKERN_IRQ_START;
7: IOKern_Interrupt_Helper( IOKERN_TASK_ID_IRQ9, IOKERN_IRQ_9, &r );
8: IOKERN_IRQ_END( IOKERN_IRQ_9 );
9: #endif
10: }
```

3.2.1.229 IOKern_PIC_EOI Function

C++

```
void IOKern_PIC_EOI(unsigned char irq);
```

File

idi.c (see page 373)

Description

brief

Body Source

```
1: void IOKern_PIC_EOI( unsigned char irq )
2: {
3: outp( IOKERN_DOS_PIC1_CMD, IOKERN_DOS_NSEOI );
4: if ( irq >= 8 ) outp( IOKERN_DOS_PIC2_CMD, IOKERN_DOS_NSEOI );
5: }
```

3.2.1.230 IOKern_Resource_Initialization Function

C++

```
void IOKern_Resource_Initialization();
```

File

idi.c (see page 373)

Body Source

```

1: void IOKern_Resource_Initialization( void )
2: {
3:     size_t     index;
4:
5:
6:
7:
8:
9:     memset( iokern_task, 0, IOKERN_TASK_QTY * sizeof( IOKERN_TASK_TYPE ) );
10:
11:    for ( index = 0; index < IOKERN_TASK_QTY; index++ )
12:    {
13:        iokern_task[index].number = IOKERN_IRQ_NONE;
14:    }
15:
16: #if defined( IOKERN_CPU_REGION_MAP )
17:
18:     index = 0;
19:     while( SYS_TYPE__NONE != iokern_cpu_region_list[index].type )
20:     {
21:         iokern_cpu_region_list[index].name = iokern_cpu_region_description[index];
22:         index++;
23:     }
24:
25:     index = 0;
26:     while( SYS_TYPE__NONE != iokern_irq_map_list[index].type )
27:     {
28:         iokern_irq_map_list[index].name = iokern_irq_map_description[index];
29:         index++;
30:     }
31: #endif
32: }
```

3.2.1.231 IOKern_Resource_Termination Function

C++

```
void IOKern_Resource_Termination();
```

File

idi.c (see page 373)

Body Source

```

1: void IOKern_Resource_Termination( void )
2: {
3:     size_t     index;
4:
5:     IOKERN_IRQ_START;
6:     for ( index = 0; index < IOKERN_TASK_QTY; index++ )
7:     {
8:         IOKern_DOS_IRQ_Free( iokern_task[index].number );
9:     }
10:    IOKERN_IRQ_ENABLE;
11: }
```

3.2.1.232 IOKern_Task_Handle_Get Function

C++

```
IOKERN_TASK_TYPE * IOKern_Task_Handle_Get(unsigned int irq);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```
1: IOKERN_TASK_TYPE * IOKern_Task_Handle_Get( unsigned int irq )
2: {
3:     size_t task_id;
4:
5:     if ( SUCCESS != IOKern_IRQ_Test( irq ) ) return( NULL );
6:
7:     task_id = 0;
8:     while( task_id < IOKERN_TASK_QTY )
9:     {
10:         if ( irq == iokern_task[task_id].number )
11:         {
12:             return( &iokern_task[task_id] );
13:         }
14:         task_id++;
15:     }
16:
17:     return NULL;
18: }
```

3.2.1.233 IOKern_Task_ID_Get Function

C++

```
IOKERN_TASK_ID_ENUM IOKern_Task_ID_Get(unsigned int irq);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: IOKERN_TASK_ID_ENUM IOKern_Task_ID_Get( unsigned int irq )
2: {
3:   IOKERN_TASK_ID_ENUM task_id;
4:
5:   if ( IOKERN_IRQ_NONE == irq )
6:   {
7:     task_id = IOKERN_TASK_ID_NONE;
8:   }
9:   else if ( irq <= IOKERN_IRQ_15 )
10:  {
11:    task_id = (IOKERN_TASK_ID_ENUM) irq;
12:  }
13:   else
14:   {
15:     task_id = IOKERN_TASK_ID_NONE;
16:   }
17:   return task_id;
18: }
```

3.2.1.234 main Function

C++

```
int main(int argc, char* argv[]);
```

File

idi.c (see page 373)

Returns

SUCCESS (0) if no errors encountered, otherwise errors are reported as a negative value.

Description

brief Processes and dispatches the top level of the command and passes the remaining string list onto specialized functions to further process arguments. If no command is specified then a help output is produced.

param[in] argc number of arguments including the executable file name param[in] argv list of string arguments lex'd from the command line

Body Source

```

1: int main( int argc, char* argv[] )
2: {
3:
4:
5:   int board_id;
6:   int error_code;
7:   int index;
8:   int argc_new;
9:   char ** argv_new;
10:  struct board_dataset * dataset;
11:
12:
13: #if defined( MAIN_DEBUG )
```

```
14: printf( "Main: Entry\n" );
15: #endif
16: #if defined( SPI_PRINT_XFR_DEBUG )
17: global_internal_tx_byte_count = 0;
18: global_internal_rx_byte_count = 0;
19: #endif
20:
21:
22:
23:
24: # if(1)
25:     setvbuf(stdout, NULL, _IONBF, 0);
26:     setvbuf(stderr, NULL, _IONBF, 0);
27: # endif
28:
29:
30: #if defined( MAIN_DEBUG )
31: printf( "Main: 1\n" );
32: #endif
33:
34: #if(0)
35: {
36:     char buf[80];
37:     int count = argc;
38:     int index = 0;
39:     while ( count > 0 )
40:     {
41:         printf( "index = %d, str = <%s>\n", index, argv[index] );
42:         index++;
43:         count--;
44:     }
45:     Time_Current_String( buf, 80 );
46:     printf( "Current time is: %s", buf );
47: }
48: #endif
49:
50:
51:     Main_Versalogic();
52:
53: #if defined( MAIN_DEBUG )
54: printf( "Main: 2\n" );
55: #endif
56:
57:     error_code      = SUCCESS;
58:     board_id        = ID_BOARD_UNKNOWN;
59:     global_board_id = board_id;
60:
61:     global_loop_space = false;
62:
63:     global_io_trace_enable = false;
64:
65:     index = 1;
66:     if ( argc > 1 )
67:     {
68:
69:
70:
71:
72:
73:
74: #if defined( MAIN_DEBUG )
75: printf( "Main: 3\n" );
76: #endif
77:
```

```

78:
79:     argv_new    = &(argv[index]);
80:     argc_new   = argc - index;
81:     error_code = Command_Line_Prefix( argc_new, argv_new );
82:     if ( error_code < 0 ) goto Main_Termination_Skip_Termination;
83:     else
84:         index += error_code;
85:
86:
87:
88:     switch( Command_Line_Main_Board_Type_Status( argv[index], &board_id ) )
89:     {
90:         case CLM_BTS_NORMAL:
91:             break;
92:         case CLM_BTS IMPLIED:
93:             global_board_id = board_id;
94:             goto Main_Initialization_Start;
95:         case CLM_BTS_NOT_REQUIRED:
96:             goto Main_Loop_Start;
97:     }
98:
99:
100:    if      ( 0 == strcmpi( "idi", argv[index] ) )
101:    {
102:        board_id    = ID_IDI48;
103:        global_board_id = board_id;
104:        index++;
105:    }
106:    else if   ( 0 == strcmpi( "ido", argv[index] ) )
107:    {
108:        board_id      = ID_IDO48;
109:        global_board_id = board_id;
110:        index++;
111:    }
112:    else
113:    {
114:        error_code = -EC_BOARD_TYPE;
115:        goto Main_Termination_Error_Codes_Skip_Termination;
116:    }
117:
118:
119: Main_Initialization_Start:
120: #if defined( MAIN_DEBUG )
121: printf( "Main:  Main_Initialization_Start\n" );
122: #endif
123:     error_code = Initialization( board_id );
124:     if ( SUCCESS != error_code ) goto Main_Termination;
125:
126:     dataset      = ( struct board_dataset * ) board_definition[board_id].dataset;
127:
128:
129: Main_Loop_Start:
130: #if defined( MAIN_DEBUG )
131: printf( "Main:  Main_Loop_Start\n" );
132: #endif
133: do
134: {
135:     argv_new    = &(argv[index]);
136:     argc_new   = argc - index;
137:     error_code = Command_Line_Main( board_id, argc_new, argv_new );
138:     if ( SUCCESS != error_code ) goto Main_Termination_Error_Codes;
139:
140:     switch( board_id )
141:     {

```

```
142:     case ID_IDI48:
143:         if ( -1 == IDI_Main_Loop_Testing( struct idi_dataset * ) dataset )
144:         {
145:             goto Main_Termination;
146:         }
147:         break;
148:     }
149:
150:     if ( global_loop_command )
151:     {
152:         if ( global_loop_delay_ms > 0 )
153:         {
154:             #if defined( __MSDOS__ )
155:                 delay( global_loop_delay_ms );
156:             #endif
157:         }
158:
159:         if ( global_loop_count > 0 )
160:         {
161:             global_loop_count_counter--;
162:             if ( 0 == global_loop_count_counter ) global_loop_command = false;
163:         }
164:
165:         if ( true == global_loop_space )
166:         {
167:             int character;
168:             BOOL not_done;
169:
170:             not_done = true;
171:             do
172:             {
173:                 if ( Character_Get( &character ) )
174:                 {
175:                     if ( ' ' == ( char ) character )
176:                     {
177:                         not_done = false;
178:                     }
179:                     else
180:                     {
181:                         not_done = false;
182:                         global_loop_command = false;
183:                     }
184:                 }
185:             } while( true == not_done );
186:         }
187:         else
188:         {
189:             if ( Character_Get( NULL ) ) global_loop_command = false;
190:         }
191:
192:
193:     }
194:
195: } while ( global_loop_command );
196: }
197: else
198: {
199:     Help( false , stdout );
200:     goto Main_Termination_Skip_Termination;
201: }
202:
203: Main_Termination:
204:
205: #if defined( MAIN_DEBUG )
```

```

206: printf( "Main: Main_Termination\n" );
207: #endif
208:     Termination( board_id );
209:
210: Main_Termination_Skip_Termination:
211:
212: #if defined( MAIN_DEBUG )
213: printf( "Main: Main_Termination_Skip_Termination\n" );
214: #endif
215:
216:     return error_code;
217:
218: Main_Termination_Error_Codes:
219:
220:     Termination( board_id );
221:
222: Main_Termination_Error_Codes_Skip_Termination:
223:     printf( "ERROR: %d, %s\n", error_code, EC_Code_To_Human_Readable( error_code ) );
224:     return error_code;
225: }
```

3.2.1.235 Main_Versalogic Function

C++

```
static void Main_Versalogic();
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static void Main_Versalogic( void )
2: {
3:
4: #if defined( __MSDOS__ )
5:
6:     outportb( 0x00E2, 2 );
7: #endif
8: }
```

3.2.1.236 Print_Byte_List Function

C++

```
static int Print_Byte_List(const char * prefix_string, int argc, char ** argv, size_t
list_count, uint8_t * list, FILE * out);
```

File

idi.c (see page 373)

Parameters

Parameters	Description
const char * prefix_string	beginning of line output
int argc	mode list count
char ** argv	"binary", "group" or "hex" is valid
size_t list_count	number of bytes in the list
uint8_t * list	list of bytes to be output/displayed
FILE * out	destination of the list

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

Body Source

```

1: static int Print_Byte_List( const char * prefix_string,
2:                           int      argc,
3:                           char **  argv,
4:                           size_t   list_count,
5:                           uint8_t * list,
6:                           FILE *   out
7:                           )
8: {
9:     enum { MODE_NONE = 0, MODE_BINARY = 1, MODE_HEX = 2, MODE_ALL = 3 } mode_out;
10:    int group;
11:
12:    mode_out = MODE_ALL;
13:    if ( NULL == argv )
14:    {
15:        mode_out = MODE_ALL;
16:    }
17:    else
18:    {
19:        int index;
20:        for ( index = 0; index < argc; index++ )
21:        {
22:            if ( 0 == strcmpi( "binary", argv[index] ) ) mode_out |= MODE_BINARY;
23:            else if ( 0 == strcmpi( "group", argv[index] ) ) mode_out |= MODE_HEX;
24:            else if ( 0 == strcmpi( "hex", argv[index] ) ) mode_out |= MODE_HEX;
25:            else
26:                mode_out |= MODE_ALL;
27:        }
28:
29:        if ( MODE_BINARY == ( mode_out & MODE_BINARY ) )
30:        {
31:            int cp;
32:            int channel;
33:            uint8_t mask;
34:            char message[64];
35:
36:            cp = 0;
37:            group = 0;
38:            for ( group = 0; group < list_count; group++ )
39:            {

```

```

40:     mask = 0x01;
41:     for ( channel = 0; channel < 8; channel++ )
42:     {
43:         message[cp++] = !(list[group] & mask) ? '1' : '0';
44:         mask = mask << 1;
45:     }
46:     message[cp++] = ' ';
47: }
48: message[cp] = '\0';
49: fprintf( out, "%s: %s\n", prefix_string, message );
50: }
51:
52: if ( MODE_HEX == ( mode_out & MODE_HEX ) )
53: {
54:     fprintf( out, "%s:", prefix_string );
55:     for ( group = 0; group < list_count; group++ ) fprintf( out, " %02X", list[group] );
56:     fprintf( out, "\n" );
57: }
58:
59: return SUCCESS;
60: }
```

3.2.1.237 Print_Multiple Function

C++

```
static int Print_Multiple(char * message, FILE * out);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int Print_Multiple( char * message, FILE * out )
2: {
3:     int error_code;
4:
5:     error_code = fprintf( stdout, message );
6:     if ( out != stdout )
7:     {
8:         error_code = fprintf( out, message );
9:     }
10:    return error_code;
11: }
```

3.2.1.238 Register_Acronym_To_Row Function

C++

```
static int Register_Acronym_To_Row(int board_id, const char * acronym, int * row);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static int Register_Acronym_To_Row( int board_id, const char * acronym, int * row )
2: {
3:     int index;
4:     const struct reg_definition * definition = (const struct reg_definition *)
board_definition[board_id].definition;
5:
6:     index = 0;
7:     while ( definition[index].direction != REG_DIR_NONE )
8:     {
9:         if ( 0 == strcmpi( definition[index].acronym, acronym ) )
10:        {
11:            *row = index;
12:            return SUCCESS;
13:        }
14:        index++;
15:    }
16:    return -EC_NOT_FOUND;
17: }
```

3.2.1.239 Register_Report_Csv Function

C++

```
int Register_Report_Csv(int board_id, FILE * out);
```

File

idi.c (see page 373)

Returns

SUCCESS (0) if no errors encountered, otherwise errors are reported as a negative value.

Description

brief Outputs a human readable CSV to the desired output file or stdout.

param[in] table register definition table or data structure array param[in] out destination of the human readable text to specified file or terminal.

Body Source

```

1: int Register_Report_Csv( int board_id, FILE * out )
2: {
3:     switch ( board_id )
4:     {
5:         case ID_IDI48: return IDI_Register_Report_Csv( ((const struct idi_reg_definition *)
board_definition[board_id].definition), out );
```

```

6:   case ID_IDO48: return IDO_Register_Report_CSV( ((const struct ido_reg_definition *)  
board_definition[board_id].definition), out );  
7: }  
8: return -EC_BOARD;  
9: }

```

3.2.1.240 Signal_Handler Function

C++

```
void Signal_Handler(int signal);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: void Signal_Handler( int signal )
2: {
3:   struct board_dataset * dataset = ( struct board_dataset * )
board_definition[global_board_id].dataset;
4:   (void) signal;
5:   dataset->quit_application = true;
6:   if ( true == global_io_trace_enable )
7:   {
8:     if ( IO_TRACE_STOP_CTRLC == dataset->trace.stop ) dataset->trace.active = false;
9:   }
10: }

```

3.2.1.241 SPI_Calculate_Clock Function

C++

```
int SPI_Calculate_Clock(double clock_request_hz, double * clock_actual_hz, double * error,  
uint16_t * hci);
```

File

idi.c (see page 373)

Parameters

Parameters	Description
double clock_request_hz	requested spi clock frequency
double * clock_actual_hz	computed actual clock frequency
double * error	error between actual and desired
uint16_t * hci	computed count

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Computes the SPI clock half clock register value given a requested SPI clock frequency. It will also produce a 'report' indicating the actual value (due to integer resolution) as well as a computed error between requested and actual. The eror can be used to determine whether timing constraints are met.

param[in] `clock_request_hz` Request clock frequency in Hertz. Example: 1.0e6 is 1MHz. **param[in]** `clock_actual_hz` Actual computed frequency. If this pointer is NULL, then it is not output. **param[out]** `error` Error between requested and actual. If this pointer is NULL, then it is not output. **param[out]** `hci` Half clock register value computed. If this pointer is NULL, then it is not output.

Body Source

```

1: int SPI_Calculate_Clock( double      clock_request_hz,
2:                           double *    clock_actual_hz,
3:                           double *    error,
4:                           uint16_t *  hci
5:                         )
6: {
7:     int   error_code;
8:     double half_clock_request_sec;
9:     double half_clock_actual_sec;
10:    double error_internal;
11:    double scratch;
12:    uint16_t hci_internal;
13:
14:    half_clock_request_sec = 1.0 / ( 2.0 * clock_request_hz );
15:
16:    error_code = SPI_Calculate_Half_Clock( half_clock_request_sec,
17:                                           &half_clock_actual_sec,
18:                                           &error_internal,
19:                                           &hci_internal
20:                                         );
21:    if ( SUCCESS != error_code ) return error_code;
22:
23:
24:    scratch = 1.0 / ( 2.0 * half_clock_actual_sec );
25:    if ( NULL != error ) *error = ( scratch - clock_request_hz ) /
26:        clock_request_hz;
27:    if ( NULL != clock_actual_hz ) *clock_actual_hz = scratch;
28:    if ( NULL != hci ) *hci = (uint16_t) hci_internal;
29:    return SUCCESS;
30: }
```

3.2.1.242 SPI_Calculate_End_Cycle_Delay Function

C++

```
int SPI_Calculate_End_Cycle_Delay(double spi_half_clock_interval_sec, double
delay_request_sec, double * delay_actual_sec, double * error, uint8_t * ecd);
```

File

idi.c (see page 373)

Parameters

Parameters	Description
double spi_half_clock_interval_sec	calculated half-clock interval
double delay_request_sec	requested end-delay interval
double * delay_actual_sec	computed actual delay
double * error	error between actual and desired
uint8_t * ecd	computed count

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Computes the time delay at the end of each byte transmitted. It will only output the parameters whose pointers are not NULL.

param[in] spi_half_clock_interval_sec Computed half clock interval in seconds param[in] delay_request_sec Requested time delay in seconds param[out] delay_actual_sec Pointer to actual time delay computed, if not NULL. param[out] error Pointer to error value computed, if not NULL. param[out] ecd Pointer to the computed end-cycle-delay, if not NULL.

Body Source

```

1: int SPI_Calculate_End_Cycle_Delay( double      spi_half_clock_interval_sec,
2:                                     double      delay_request_sec,
3:                                     double *   delay_actual_sec,
4:                                     double *   error,
5:                                     uint8_t *  ecd
6:                                     )
7: {
8:
9:     double      scratch;
10:    int       ecd_temp;
11:
12:
13:    scratch = ( delay_request_sec - 4.0 * CLOCK_PERIOD_SEC ) / spi_half_clock_interval_sec;
14:    ecd_temp = (int) scratch;
15:
16:    if ( ( ecd_temp > 255 ) || ( ecd_temp < 0 ) ) return -EC_SPI_ECD_OUT_OF_RANGE;
17:
18:
19:    scratch = CLOCK_PERIOD_SEC * 4.0 + ((double) ecd_temp) * spi_half_clock_interval_sec;
20:    if ( NULL != error ) *error = ( scratch - delay_request_sec ) /
delay_request_sec;
21:    if ( NULL != delay_actual_sec ) *delay_actual_sec = scratch;
22:    if ( NULL != ecd ) *ecd = (uint8_t) ecd_temp;
23:
24:    return SUCCESS;
25: }
```

3.2.1.243 SPI_Calculate_Half_Clock Function

C++

```
int SPI_Calculate_Half_Clock(double half_clock_request_sec, double * half_clock_actual_sec,
double * error, uint16_t * hci);
```

File

idi.c (see page 373)

Parameters

Parameters	Description
double half_clock_request_sec	requested half clock interval
double * half_clock_actual_sec	computed actual half clock interval
double * error	error between actual and desired
uint16_t * hci	computed count

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Computes the half clock register value given a requested time interval. It will also produce a 'report' indicating the actual value (due to integer resolution) as well as a computed error between requested and actual. The error can be used to determine whether timing constraints are met.

param[in] half_clock_request_sec Request time interval in seconds. Example: 20.0e-6 is 20uS. param[in] half_clock_actual_sec Actual computed time. If this pointer is NULL, then it is not output. param[out] error Error between requested and actual. If this pointer is NULL, then it is not output. param[out] hci Half clock register value computed. If this pointer is NULL, then it is not output.

Body Source

```

1: int SPI_Calculate_Half_Clock( double      half_clock_request_sec,
2:                               double *    half_clock_actual_sec,
3:                               double *    error,
4:                               uint16_t *  hci
5:                           )
6: {
7:     double      scratch;
8:     int        hci_temp;
9:
10:
11:    scratch = ( half_clock_request_sec / CLOCK_PERIOD_SEC ) - 4.0;
12:    hci_temp = (int) scratch;
13:
14:    if ( ( hci_temp > 4095 ) || ( hci_temp < 0 ) ) return -EC_SPI_HALF_CLOCK_OUT_OF_RANGE;
15:
16:
17:    scratch = CLOCK_PERIOD_SEC * ( 4.0 + ((double) hci_temp) );
18:    if ( NULL != error ) *error = ( scratch -
half_clock_request_sec ) / half_clock_request_sec;
19:    if ( NULL != half_clock_actual_sec ) *half_clock_actual_sec = scratch;
20:    if ( NULL != hci ) *hci = (uint16_t) hci_temp;
21:
22:    return SUCCESS;
23: }
```

3.2.1.244 SPI_Calculate_Half_Clock_Interval_Sec Function

C++

```
double SPI_Calculate_Half_Clock_Interval_Sec(uint16_t half_clock_interval);
```

File

idi.c (see page 373)

Parameters

Parameters	Description
uint16_t half_clock_interval	hci

Returns

The time value as a double in units of seconds.

Description

brief Computes the half clock interval in seconds given the value from the half clock interval register.

param[in] half_clock_interval Half clock interval register value

Body Source

```
1: double SPI_Calculate_Half_Clock_Interval_Sec( uint16_t half_clock_interval )
2: {
3:     double half_clock_interval_sec;
4:     half_clock_interval_sec = CLOCK_PERIOD_SEC * ( 4.0 + ((double) half_clock_interval ) );
5:     return half_clock_interval_sec;
6: }
```

3.2.1.245 SPI_Commit Function

C++

```
int SPI_Commit(int board_id, uint8_t commit);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Sets/Clears the chip select or used to commit the transmit/write FIFO to the spi interface. The mode of operation is dependent on the chip_select_behavior.

param[in] commit Used to write to the SCS_COMMIT bit. Its behavior is dependent on the chip_select_behavior.

Body Source

```

1: int SPI_Commit( int board_id, uint8_t commit )
2: {
3: #define FUNCTION_NAME__ "SPI_Commit"
4: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
5: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
6: #endif
7: #undef FUNCTION_NAME__
8:
9: if ( SPI_IsNotPresent( board_id ) ) return -EC_SPI_NOT_FOUND;
10: commit = commit & SPI_COMMIT_BIT_POSITION;
11: switch ( board_id )
12: {
13: case ID_IDI48: IO_Write_U8( board_id, __LINE__, IDI_SPI_COMMIT, commit ); break;
14: case ID_IDO48: IO_Write_U8( board_id, __LINE__, IDO_SPI_COMMIT, commit ); break;
15: }
16: return SUCCESS;
17: }
```

3.2.1.246 SPI_Commit_Get Function

C++

```
int SPI_Commit_Get(int board_id, uint8_t * commit);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Sets/Clears the chip select or used to commit the transmit/write FIFO to the spi interface. The mode of operation is dependent on the chip_select_behavior.

param[out] commit Used to write to the SCS_COMMIT bit. Its behavior is dependent on the chip_select_behavior.

Body Source

```

1: int SPI_Commit_Get( int board_id, uint8_t * commit )
2: {
3: uint8_t value;
4:
5: #define FUNCTION_NAME__ "SPI_Commit_Get"
6: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
7: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
8: #endif
9: #undef FUNCTION_NAME__
10:
11:
12: if ( SPI_IsNotPresent( board_id ) ) return -EC_SPI_NOT_FOUND;
13: switch ( board_id )
14: {
15: case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_COMMIT, &value ); break;
16: case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_COMMIT, &value ); break;
```

```

17: }
18: *commit = value & SPI_COMMIT_BIT_POSITION;
19: return SUCCESS;
20: }
```

3.2.1.247 SPI_Commit_Is_Inactive Function

C++

```
BOOL SPI_Commit_Is_Inactive(int board_id);
```

File

idi.c ([see page 373](#))

Returns

true if commit is inactive. In other words, if the SPI state machine has completed the data transfer including any chip select changes that might occur at the end of buffer ([see page 325](#)).

Description

brief

Body Source

```

1: BOOL SPI_Commit_Is_Inactive( int board_id )
2: {
3:     uint8_t value;
4:
5: #define FUNCTION_NAME__ "SPI_Commit_Is_Inactive"
6: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
7:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
8: #endif
9: #undef FUNCTION_NAME__
10:
11:    switch ( board_id )
12:    {
13:        case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_COMMIT, &value ); break;
14:        case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_COMMIT, &value ); break;
15:    }
16:
17:    if ( 0 == (value & SPI_COMMIT_BIT_POSITION) ) return true;
18:    return false;
19: }
```

3.2.1.248 SPI_Configuration_Chip_Select_Behavior_Get Function

C++

```
int SPI_Configuration_Chip_Select_Behavior_Get(int board_id, SPI_CSBE_NUM *  
chip_select_behavior);
```

File

idi.c ([see page 373](#))

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Extracts the chip select behavior from the SPI configuration register.

param[out] chip_select_behavior pointer to the destination of the value obtained.

Body Source

```

1: int SPI_Configuration_Chip_Select_Behavior_Get( int board_id, SPI_CSB_ENUM *
chip_select_behavior )
2: {
3:     uint8_t     scratch;
4:
5: #define FUNCTION_NAME__ "SPI_Configuration_Chip_Select_Behavior_Get"
6: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
7:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
8: #endif
9: #undef FUNCTION_NAME__
10:
11: if ( SPI_IsNotPresent( board_id ) ) return -EC_SPI_NOT_FOUND;
12:
13: switch ( board_id )
14: {
15:     case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_CONFIG, &scratch ); break;
16:     case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_CONFIG, &scratch ); break;
17: }
18:
19: *chip_select_behavior = (SPI_CSB_ENUM) ( scratch >> 4 ) & 0x03;
20: return SUCCESS;
21: }
```

3.2.1.249 SPI_Configuration_Chip_Select_Behavior_Set Function

C++

```
int SPI_Configuration_Chip_Select_Behavior_Set(int board_id, SPI_CSB_ENUM
chip_select_behavior);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Sets the chip select behavior to the SPI configuration register.

param[in] chip_select_behavior enumerated value to be written to the register.

Body Source

```

1: int SPI_Configuration_Chip_Select_Behavior_Set( int board_id, SPI_CSBOARD_ENUM
chip_select_behavior )
2: {
3:     uint8_t     scratch;
4:     int         spi_config_symbol;
5:
6: #define FUNCTION_NAME__ "SPI_Configuration_Chip_Select_Behavior_Set"
7: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
8:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
9: #endif
10: #undef FUNCTION_NAME__
11:
12: if ( SPI_IsNotPresent( board_id ) ) return -EC_SPI_NOT_FOUND;
13:
14: switch ( board_id )
15: {
16:     case ID_IDI48: spi_config_symbol = (int) IDI_SPI_CONFIG; break;
17:     case ID_IDO48: spi_config_symbol = (int) IDO_SPI_CONFIG; break;
18: }
19:
20: IO_Read_U8( board_id, __LINE__, spi_config_symbol, &scratch );
21:
22: scratch &= ~0x30;
23: scratch |= (uint8_t) ( ( chip_select_behavior & 0x03 ) << 4 );
24:
25: IO_Write_U8( board_id, __LINE__, spi_config_symbol, scratch );
26: return SUCCESS;
27: }
```

3.2.1.250 SPI_Configuration_Chip_Select_Route_Get Function

C++

```
int SPI_Configuration_Chip_Select_Route_Get(int board_id, BOOL * cs_to_channel1);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Sets which hardware chip select line is to be used, either CS0 (channel 0) or CS1 (channel 1).

param[out] chip_select_behavior pointer to the destination of the value obtained.

Body Source

```

1: int SPI_Configuration_Chip_Select_Route_Get( int board_id, BOOL * cs_to_channel1 )
2: {
3:     uint8_t     scratch;
4:     uint8_t     mask = 0x40;
5:
6: #define FUNCTION_NAME__ "SPI_Configuration_Chip_Select_Route_Get"
```

```

7: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
8: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
9: #endif
10: #undef FUNCTION_NAME__
11:
12: if ( SPI_IsNotPresent( board_id ) ) return -EC_SPI_NOT_FOUND;
13:
14: switch ( board_id )
15: {
16:   case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_CONFIG, &scratch ); break;
17:   case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_CONFIG, &scratch ); break;
18: }
19:
20: if ( scratch & mask ) *cs_to_channel1 = true;
21: else                  *cs_to_channel1 = false;
22:
23: return SUCCESS;
24: }
```

3.2.1.251 SPI_Configuration_Chip_Select_Route_Set Function

C++

```
int SPI_Configuration_Chip_Select_Route_Set(int board_id, BOOL cs_to_channel1);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Sets which hardware chip select line is to be used, either CS0 (channel 0) or CS1 (channel 1).

param[in] cs_to_channel1 true if CS to route to channel 1 (CS1), otherwise channel 0 (CS0).

Body Source

```

1: int SPI_Configuration_Chip_Select_Route_Set( int board_id, BOOL cs_to_channel1 )
2: {
3:   int spi_config_symbol;
4:   uint8_t scratch;
5:   uint8_t mask = 0x40;
6:
7: #define FUNCTION_NAME__ "SPI_Configuration_Chip_Select_Route_Set"
8: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
9: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
10: #endif
11: #undef FUNCTION_NAME__
12:
13: if ( SPI_IsNotPresent( board_id ) ) return -EC_SPI_NOT_FOUND;
14:
15: switch ( board_id )
16: {
17:   case ID_IDI48: spi_config_symbol = (int) IDI_SPI_CONFIG; break;
18:   case ID_IDO48: spi_config_symbol = (int) IDO_SPI_CONFIG; break;
19: }
```

```

20:
21: IO_Read_U8( board_id, __LINE__, spi_config_symbol, &scratch );
22:
23: if ( true == cs_to_channel1 ) scratch |= mask;
24: else
25:     scratch &= ~mask;
26: IO_Write_U8( board_id, __LINE__, spi_config_symbol, scratch );
27: return SUCCESS;
28: }
```

3.2.1.252 SPI_Configuration_Get Function

C++

```
int SPI_Configuration_Get(int board_id, struct spi_cfg * cfg);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Obtains the SPI configuration from the hardware.

param[out] cfg SPI configuration data structure or data set

Body Source

```

1: int SPI_Configuration_Get( int board_id, struct spi_cfg * cfg )
2: {
3:     uint8_t     scratch;
4:     struct board_dataset * dataset      = ( struct board_dataset * )
board_definition[board_id].dataset;
5:
6: #define FUNCTION_NAME__ "SPI_Configuration_Get"
7: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
8: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
9: #endif
10: #undef FUNCTION_NAME__
11:
12: if ( SPI_IsNotPresent( board_id ) ) return -EC_SPI_NOT_FOUND;
13:
14: switch ( board_id )
15: {
16:     case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_CONFIG, &scratch ); break;
17:     case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_CONFIG, &scratch ); break;
18: }
19:
20:
21: cfg->chip_select_route_ch1  = ( scratch & 0x40 ) ? true : false;
22: cfg->chip_select_behavior  = (SPI_CSB_ENUM) ( scratch >> 4 ) & 0x03;
23: cfg->sclk_polarity        = ( scratch & 0x01 ) ? true : false;
24: cfg->sclk_phase           = ( scratch & 0x02 ) ? true : false;
25: cfg->sdi_polarity          = ( scratch & 0x04 ) ? true : false;
26: cfg->sdo_polarity          = ( scratch & 0x08 ) ? true : false;
27: cfg->sdio_wrap             = ( scratch & 0x80 ) ? true : false;
```

```

28:
29:     switch ( board_id )
30:     {
31:         case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_HCI_LSB, &scratch ); break;
32:         case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_HCI_LSB, &scratch ); break;
33:     }
34:
35:     cfg->half_clock_interval = (uint16_t) scratch;
36:     switch ( board_id )
37:     {
38:         case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_HCI_MSB, &scratch ); break;
39:         case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_HCI_MSB, &scratch ); break;
40:     }
41:
42:     cfg->half_clock_interval |= ( (uint16_t) scratch) << 8;
43:     switch ( board_id )
44:     {
45:         case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_ECD, &(cfg->end_cycle_delay) );
break;
46:         case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_ECD, &(cfg->end_cycle_delay) );
break;
47:     }
48:
49:
50:     cfg->clock_hz = 1.0 / ( 2.0 * CLOCK_PERIOD_SEC * ( 4.0 + ((double)
cfg->half_clock_interval) ) );
51:     cfg->end_delay_ns = 1.0e9 * CLOCK_PERIOD_SEC * 4.0 + 0.5 * ((double)
cfg->end_cycle_delay) / cfg->clock_hz;
52:
53:     if ( cfg != &(dataset->spi.cfg) )
54:     {
55:         memcpy( &(dataset->spi.cfg), &cfg, sizeof( struct spi_cfg ) );
56:     }
57:     return SUCCESS;
58: }
```

3.2.1.253 SPI_Configuration_Initialize Function

C++

```
int SPI_Configuration_Initialize(struct spi_cfg * cfg);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Initializes the SPI configuration data structure

parma[in] cfg Pointer to the SPI configuration data structure to be initialized

Body Source

```

1: int SPI_Configuration_Initialize( struct spi_cfg * cfg )
2: {
```

```

3:   cfg->sdio_wrap      = false;
4:   cfg->sdo_polarity    = false;
5:   cfg->sdi_polarity    = false;
6:
12:  cfg->sclk_phase     = false;
13:  cfg->sclk_polarity   = false;
14:  cfg->chip_select_route_ch1 = false;
15:  cfg->chip_select_behavior = CSB_SOFTWARE;
16:  cfg->end_cycle_delay  = 0;
17:  cfg->half_clock_interval = 0;
18:
19:  cfg->clock_hz        = 0.0;
20:  cfg->end_delay_ns    = 0.0;
21:
22:  return SUCCESS;
23: }
```

3.2.1.254 SPI_Configuration_Set Function

C++

```
int SPI_Configuration_Set(int board_id, struct spi_cfg * cfg);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Commits the configuration data structure to the hardware.

param[in] cfg The software configuration data structure to be committed to hardware

Body Source

```

1: int SPI_Configuration_Set( int board_id, struct spi_cfg * cfg )
2: {
3:   int error_code;
4:   double scratch;
5:   double hci_sec;
6:   uint8_t config;
7:   struct board_dataset * dataset = ( struct board_dataset * )
board_definition[board_id].dataset;
8:
9: #define FUNCTION_NAME__ "SPI_Configuration_Set"
10: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
11:   IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
12: #endif
13: #undef FUNCTION_NAME__
14:
15:   if ( SPI_IsNotPresent( board_id ) ) return -EC_SPI_NOT_FOUND;
16:
17:   config = (uint8_t) ( ( cfg->chip_select_behavior & 0x03 ) << 4 );
18:   if ( cfg->sclk_polarity ) config |= 0x01;
19:   if ( cfg->sclk_phase ) config |= 0x02;
20:   if ( cfg->sdi_polarity ) config |= 0x04;
```

```
21: if ( cfg->sdo_polarity ) config |= 0x08;
22: if ( cfg->sdio_wrap ) config |= 0x80;
23: if ( cfg->chip_select_route_ch1 ) config |= 0x40;
24:
25: switch ( board_id )
26: {
27: case ID_IDI48: IO_Write_U8( board_id, __LINE__, IDI_SPI_CONFIG, config ); break;
28: case ID_IDO48: IO_Write_U8( board_id, __LINE__, IDO_SPI_CONFIG, config ); break;
29: }
30:
31:
32: if ( cfg->clock_hz > 0 )
33: {
34:
35: error_code = SPI_Calculate_Clock( cfg->clock_hz, NULL, NULL, &(cfg->half_clock_interval) );
36: if ( SUCCESS != error_code ) return error_code;
37: }
38: hci_sec = SPI_Calculate_Half_Clock_Interval_Sec( cfg->half_clock_interval );
39:
40: if ( cfg->end_delay_ns > 0 )
41: {
42: scratch = cfg->end_delay_ns * 1.0e-9;
43: error_code = SPI_Calculate_End_Cycle_Delay( hci_sec,
44:                                             scratch,
45:                                             NULL,
46:                                             NULL,
47:                                             &(cfg->end_cycle_delay)
48: );
49: if ( SUCCESS != error_code ) return error_code;
50: }
51:
52: switch ( board_id )
53: {
54: case ID_IDI48: IO_Write_U8( board_id, __LINE__, IDI_SPI_HCI_LSB, (uint8_t)(
cfg->half_clock_interval & 0xFF ) ); break;
55: case ID_IDO48: IO_Write_U8( board_id, __LINE__, IDO_SPI_HCI_LSB, (uint8_t)(
cfg->half_clock_interval & 0xFF ) ); break;
56: }
57:
58:
59:
60: switch ( board_id )
61: {
62: case ID_IDI48: IO_Write_U8( board_id, __LINE__, IDI_SPI_HCI_MSB, (uint8_t)(
cfg->half_clock_interval >> 8 ) ); break;
63: case ID_IDO48: IO_Write_U8( board_id, __LINE__, IDO_SPI_HCI_MSB, (uint8_t)(
cfg->half_clock_interval >> 8 ) ); break;
64: }
65:
66:
67: switch ( board_id )
68: {
69: case ID_IDI48: IO_Write_U8( board_id, __LINE__, IDI_SPI_ECD, cfg->end_cycle_delay );
break;
70: case ID_IDO48: IO_Write_U8( board_id, __LINE__, IDO_SPI_ECD, cfg->end_cycle_delay );
break;
71: }
72:
73:
74: if ( cfg != &(dataset->spi.cfg) )
75: {
76: memcpy( &(dataset->spi.cfg), &cfg, sizeof( struct spi_cfg ) );
77: }
```

```

78:     return SUCCESS;
79: }

```

3.2.1.255 SPI_Data_Read Function

Special case of Write/Read that has a function signature same as fread() or fwrite().

param[in] cfg pass in the configuration to be written to hardware. return a nonzero if successful, else return zero.

**

- @brief
- @return A zero (SUCCESS) is returned if successful, otherwise a negative error
- code is returned.

C++

```
int SPI_Data_Read(int board_id, const void * rx_buffer, size_t rx_object_size, size_t
rx_object_count, FILE * fd_log);
```

File

idi.c (see page 373)

Parameters

Parameters	Description
size_t rx_object_size	*< object size
size_t rx_object_count	*< object count
FILE * fd_log	*< set to NULL if no file logging

Body Source

```

1: int SPI_Data_Read( int board_id,
2:                     const void * rx_buffer,
3:                     size_t rx_object_size,
4:                     size_t rx_object_count,
5:                     FILE * fd_log
6:                     )
7: {
8:     int error_code;
9:
10: #define FUNCTION_NAME__ "SPI_Data_Read"
11: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
12:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
13: #endif
14: #undef FUNCTION_NAME__
15:
16:     error_code = SPI_Data_Write_Read( board_id,
17:                                         rx_object_size,
18:                                         0,
19:                                         NULL,
20:                                         rx_object_count,
21:                                         rx_buffer

```

```

22:         );
23:     if ( SUCCESS != error_code ) goto SPI_Data_Read_Error_Exit;
24:
25:     if ( NULL != fd_log )
26:     {
27:         error_code = fwrite( rx_buffer, rx_object_size, rx_object_count, fd_log );
28:     }
29:     if ( error_code < 0 ) goto SPI_Data_Read_Error_Exit;
30:
31:     return SUCCESS;
32:
33: SPI_Data_Read_Error_Exit:
34:
35: #define FUNCTION_NAME__ "SPI_Data_Read"
36: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
37:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, error_code );
38: #endif
39: #undef FUNCTION_NAME__
40:
41:     return error_code;
42: }
```

3.2.1.256 SPI_Data_Write Function

Special case of Write/Read that has a function signature same as fread() or fwrite().

param[in] cfg pass in the configuration to be written to hardware. return a nonzero if successful, else return zero.

* *

- @brief
- @return A zero (SUCCESS) is returned if successful, otherwise a negative error
- code is returned.

C++

```
int SPI_Data_Write(int board_id, const void * tx_buffer, size_t tx_object_size, size_t
tx_object_count, FILE * fd_log);
```

File

idi.c (see page 373)

Parameters

Parameters	Description
size_t tx_object_size	*< object size
size_t tx_object_count	*< object count
FILE * fd_log	*< set to NULL if no file logging

Body Source

```

1: int SPI_Data_Write( int    board_id,
2:                      const void *   tx_buffer,
```

```

3:     size_t      tx_object_size,
4:     size_t      tx_object_count,
5:     FILE *      fd_log
6:     )
7: {
8:     int error_code;
9:
10: #define FUNCTION_NAME__ "SPI_Data_Write"
11: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
12:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
13: #endif
14: #undef FUNCTION_NAME__
15:
16:     error_code = SPI_Data_Write_Read( board_id,
17:                                         tx_object_size,
18:                                         tx_object_count,
19:                                         tx_buffer,
20:                                         0,
21:                                         NULL
22:                                         );
23:     if ( SUCCESS != error_code ) goto SPI_Data_Write_Error_Exit;
24:
25:     if ( NULL != fd_log )
26:     {
27:         error_code = fwrite( tx_buffer, tx_object_size, tx_object_count, fd_log );
28:     }
29:     if ( error_code < 0 ) goto SPI_Data_Write_Error_Exit;
30:
31:     return SUCCESS;
32:
33: SPI_Data_Write_Error_Exit:
34:
35: #define FUNCTION_NAME__ "SPI_Data_Write"
36: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
37:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, error_code );
38: #endif
39: #undef FUNCTION_NAME__
40:
41:     return error_code;
42: }
```

3.2.1.257 SPI_Data_Write_Read Function

C++

```
int SPI_Data_Write_Read(int board_id, size_t object_size, size_t tx_object_count, const void * tx_buffer, size_t rx_object_count, const void * rx_buffer);
```

File

idi.c (see page 373)

Parameters

Parameters	Description
size_t object_size	*< TX & RX object size: 1 <= object_size <= FIFO_SIZE
size_t tx_object_count	*< object count
size_t rx_object_count	*< object count

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

If CSB = software. Stuff as fast as you can.

If CSB == buffer (see page 325). if object size = 1, then stuff as fast as you can. The chip select will go active and only go inactive when done transmitting. This is best used for objects whos size is less than SPI_FIFO_SIZE. if object size != 1, then stuff only that quantity and wait until buffers are empty in order to have the chip select wrap around the data.

If CSB == uint8_t (see page 321) Stuff as fast as you can.

If CSB == uint16_t (see page 321) Stuff two bytes at a time as fast as you can. The object_size must be set to 2 for this to work.

Body Source

```

1: int SPI_Data_Write_Read( int board_id,
2:                         size_t object_size,
3:                         size_t tx_object_count,
4:                         const void * tx_buffer,
5:                         size_t rx_object_count,
6:                         const void * rx_buffer
7: )
8: {
9:     int error_code;
10:    SPI_CSB_ENUM csb;
11:    size_t rx_bytes_available;
12:    BOOL rx_empty;
13:    size_t tx_bytes_available;
14:    BOOL active_tx;
15:    BOOL active_rx;
16:    BOOL tx_empty;
17:    BOOL tx_full;
18:    size_t index_tx;
19:    size_t index_rx;
20:
21:
22:
23:
24:
25:    BOOL commit_valid;
26:    BOOL commit_ready;
27:    uint8_t bit_bucket;
28:
29:
30: #if defined( SPI_PRINT_DEBUG )
31: #if defined( __MSDOS__ )
32:     static size_t internal_count = 0;
33: #endif
34: #endif
35:
36: struct board_dataset * dataset = ( struct board_dataset * )
board_definition[board_id].dataset;
37:
38: #define FUNCTION_NAME__ "SPI_Data_Write_Read"
39: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )

```

```

40: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
41: #endif
42: #undef FUNCTION_NAME__
43:
44:
45: #if defined( SPI_PRINT_DEBUG )
46: printf( "**** SPI_Data_Write_Read: Entry.\n" );
47: #endif
48: #if defined( SPI_PRINT_XFR_DEBUG )
49: printf( "**** SPI_Data_Write_Read: Entry.\n" );
50: #endif
51:
52: index_tx = 0;
53: index_rx = 0;
54:
55:
56:
57:
58: if ( ( NULL == tx_buffer ) && ( NULL == rx_buffer ) )
59: {
60:     error_code = SUCCESS;
61:     goto SPI_Data_Write_Read_Error_Exit;
62: }
63: if ( SPI_IsNotPresent( board_id ) )
64: {
65:     error_code = -EC_SPI_NOT_FOUND;
66:     goto SPI_Data_Write_Read_Error_Exit;
67: }
68:
69:
70: if ( ( NULL != tx_buffer ) && ( NULL != rx_buffer ) )
71: {
72:     if ( tx_object_count != rx_object_count )
73:     {
74:         error_code = -EC_SPI_BUFFER_MISMATCH;
75:         goto SPI_Data_Write_Read_Error_Exit;
76:     }
77: }
78: if ( NULL != tx_buffer )
79: {
80:     active_tx = true;
81: }
82: else
83: {
84:     tx_object_count = rx_object_count;
85:     active_tx      = false;
86: }
87: if ( NULL != rx_buffer )
88: {
89:     active_rx      = true;
90: }
91: else
92: {
93:     rx_object_count = tx_object_count;
94:     active_rx      = false;
95: }
96:
97:
98: SPI_Configuration_Chip_Select_Behavior_Get( board_id, &csb );
99:
100: if ( CSB_SOFTWARE == csb )
101: {
102:     uint8_t chip_select;
103:     if ( 0 == SPI_Commit_Get( board_id, &chip_select ) )

```

```
104:  {
105: #if defined( SPI_PRINT_DEBUG )
106:   printf( "SPI_Data_Write_Read: Warning Chip select not active, setting it now\n" );
107: #endif
108:   SPI_Commit( board_id, 0xFF );
109: }
110: }
111:
112: if ( CSB_uint16_t == csb )
113: {
114:   if ( object_size & 0x01 )
115:   {
116:     printf( "SPI_Data_Write_Read: Object Size is set to " );
117: #if defined( __MSDOS__ )
118:     printf( "%d", object_size );
119: #else
120:     printf( "%lu", object_size );
121: #endif
122:     printf( ", and ought to be set to 2\n" );
123:     error_code = -EC_SPI_BUFFER_SIZE_ODD;
124:     goto SPI_Data_Write_Error_Exit;
125: }
126: }
127:
128: #if defined( SPI_PRINT_DEBUG )
129: printf( "SPI_Data_Write_Read:\n" );
130: #if defined( __MSDOS__ )
131: printf( " object_size      = %d\n", object_size );
132: printf( " tx_object_count  = %d\n", tx_object_count );
133: printf( " rx_object_count  = %d\n", rx_object_count );
134: printf( " index_tx        = %d\n", index_tx );
135: printf( " index_rx        = %d\n", index_rx );
136: printf( " tx_buffer        = %p\n", tx_buffer );
137: printf( " rx_buffer        = %p\n", rx_buffer );
138: #else
139: printf( " object_size      = %lu\n", object_size );
140: printf( " tx_object_count  = %lu\n", tx_object_count );
141: printf( " rx_object_count  = %lu\n", rx_object_count );
142: printf( " index_tx        = %lu\n", index_tx );
143: printf( " index_rx        = %lu\n", index_rx );
144: printf( " tx_buffer        = %p\n", tx_buffer );
145: printf( " rx_buffer        = %p\n", rx_buffer );
146: #endif
147: printf( " active_tx       = %s\n", active_tx ? "true" : "false" );
148: printf( " active_rx       = %s\n", active_rx ? "true" : "false" );
149:
150: if ( NULL != tx_buffer )
151: {
152:   size_t count = object_size * tx_object_count;
153:   if ( count > HEX_DUMP_BYTES_PER_LINE ) count = HEX_DUMP_BYTES_PER_LINE;
154:   Hex_Dump_Line( 0, count, (uint8_t *) tx_buffer, stdout );
155: }
156: #endif
157:
158:
159: if ( CSB_BUFFER == csb )
160: {
161:   if ( object_size > SPI_FIFO_SIZE )
162:   {
163:     error_code = -EC_SPI_OBJECT_SIZE;
164:     goto SPI_Data_Write_Error_Exit;
165:   }
166: }
```

```

170:
171:
172:
173: SPI_Status_Write_FIFO_Status( board_id, &tx_full, &tx_empty, &tx_bytes_available );
174: if ( ( false == tx_empty ) && ( CSB_SOFTWARE != csb ) ) SPI_Commit( board_id, 0xFF );
175: do
176: {
177:     error_code = SUCCESS;
178:     if ( dataset->quit_application ) error_code = -EC_APP_TERMINATE_CTRLC;
179:     if ( Character_Get( NULL ) ) error_code = -EC_APP_TERMINATE;
180:     if ( SUCCESS != error_code )
181:     {
182:         printf( "SPI_Data_Write_Read initial empty tx\n" );
183:         goto SPI_Data_Write_Error_Exit;
184:     }
185:     SPI_Status_Write_FIFO_Status( board_id, &tx_full, &tx_empty, &tx_bytes_available );
186: } while ( false == tx_empty );
187:
188:
189: do
190: {
191:     error_code = SUCCESS;
192:     if ( dataset->quit_application ) error_code = -EC_APP_TERMINATE_CTRLC;
193:     if ( Character_Get( NULL ) ) error_code = -EC_APP_TERMINATE;
194:     if ( SUCCESS != error_code )
195:     {
196:         printf( "SPI_Data_Write_Read initial empty rx\n" );
197:         goto SPI_Data_Write_Error_Exit;
198:     }
199:     SPI_Status_Read_FIFO_Status( board_id, &rx_empty, &rx_bytes_available );
200:     if ( false == rx_empty )
201:     {
202:         switch ( board_id )
203:         {
204:             case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_DATA, &bit_bucket ); break;
205:             case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_DATA, &bit_bucket ); break;
206:         }
207:     }
208: } while ( false == rx_empty );
209:
210: index_tx      = 0;
211: index_rx      = 0;
212: commit_valid = false;
213: commit_ready = false;
214: while ( index_tx < tx_object_count )
215: {
216:     error_code = SUCCESS;
217:     if ( dataset->quit_application ) error_code = -EC_APP_TERMINATE_CTRLC;
218:     if ( Character_Get( NULL ) ) error_code = -EC_APP_TERMINATE;
219:     if ( SUCCESS != error_code )
220:     {
221:         printf( "SPI_Data_Write_Read main loop\n" );
222:         goto SPI_Data_Write_Error_Exit;
223:     }
224:
225: #if defined( SPI_PRINT_XFR_DEBUG )
226: # if defined( __MSDOS__ )
227:     delay( 250 );
228:
229:     SPI_Data_Write_Read_Helper_Read( board_id, object_size, tx_object_count, active_rx,
&index_rx, rx_buffer );
230:     delay( 250 );
231: # endif
232: #endif

```

```
233:
234: #if defined( SPI_PRINT_DEBUG )
235: # if defined( __MSDOS__ )
236:     internal_count++;
237:     printf( "SPI_Data_Write_Read: LOOP: internal_count = %u\n", internal_count );
238:     printf( "    object_size      = %d\n", object_size );
239:     printf( "    tx_object_count   = %d\n", tx_object_count );
240:     printf( "    rx_object_count   = %d\n", rx_object_count );
241:     printf( "    index_tx          = %d\n", index_tx );
242:     printf( "    index_rx          = %d\n", index_rx );
243:     printf( "    tx_buffer          = %p\n", tx_buffer );
244:     printf( "    rx_buffer          = %p\n", rx_buffer );
245:     delay( 1000 );
246: # endif
247: #endif
248:
249:
252:     SPI_Data_Write_Read_Helper_Read( board_id, object_size, tx_object_count, active_rx,
&index_rx, rx_buffer );
253:
254:
255: #if defined( SPI_PRINT_DEBUG )
256: # if defined( __MSDOS__ )
257:     if ( index_rx != index_tx )
258:     {
259:         error_code = -EC_SPI_BUFFER_MISMATCH;
260:         printf( "SPI_Data_Write_Read: MISMATCH: internal_count = %u\n", internal_count );
261:
262: #if defined( __MSDOS__ )
263:         printf( "    object_size      = %d\n", object_size );
264:         printf( "    tx_object_count   = %d\n", tx_object_count );
265:         printf( "    rx_object_count   = %d\n", rx_object_count );
266:         printf( "    index_tx          = %d\n", index_tx );
267:         printf( "    index_rx          = %d\n", index_rx );
268:         printf( "    tx_buffer          = %p\n", tx_buffer );
269:         printf( "    rx_buffer          = %p\n", rx_buffer );
270: #else
271:         printf( "    object_size      = %lu\n", object_size );
272:         printf( "    tx_object_count   = %lu\n", tx_object_count );
273:         printf( "    rx_object_count   = %lu\n", rx_object_count );
274:         printf( "    index_tx          = %lu\n", index_tx );
275:         printf( "    index_rx          = %lu\n", index_rx );
276:         printf( "    tx_buffer          = %p\n", tx_buffer );
277:         printf( "    rx_buffer          = %p\n", rx_buffer );
278: #endif
279:         printf( "    active_tx          = %s\n", active_tx ? "true" : "false" );
280:         printf( "    active_rx          = %s\n", active_rx ? "true" : "false" );
281:
282:         goto SPI_Data_Write_Read_Error_Exit;
283:     }
284: # endif
285: #endif
286:
287:
288:     SPI_Data_Write_Read_Helper_Commit( board_id, object_size, csb, commit_ready,
commit_valid );
289:
290:
293:     SPI_Data_Write_Read_Helper_Write( board_id,
294:         object_size,
295:         tx_object_count,
296:         csb,
297:         active_tx,
298:         commit_valid,
```

```

299:             &commit_ready,
300:             &index_tx,
301:             tx_buffer
302:         );
303:
304:
305:
306:
307:     SPI_Data_Write_Read_Helper_Commit( board_id, object_size, csb, commit_ready,
308:     &commit_valid );
309: }
310: #if defined( SPI_PRINT_DEBUG )
311: printf( " SPI_Data_Write_Read: Post write operations\n" );
312: #endif
313:
314:
315: do
316: {
317:     error_code = SUCCESS;
318:     if ( dataset->quit_application ) error_code = -EC_APP_TERMINATE_CTRLC;
319:     if ( Character_Get( NULL ) ) error_code = -EC_APP_TERMINATE;
320:     if ( SUCCESS != error_code )
321:     {
322:         printf( "SPI_Data_Write_Read status ending portion\n" );
323:         goto SPI_Data_Write_Error_Exit;
324:     }
325:     SPI_Data_Write_Read_Helper_Commit( board_id, object_size, csb, commit_ready,
326:     &commit_valid );
327:     SPI_Status_Write_FIFO_Status( board_id, &tx_full, &tx_empty, &tx_bytes_available );
328:
329: #if defined( SPI_PRINT_DEBUG )
330: printf( " SPI_Data_Write_Read: Post Read Begin\n" );
331: #endif
332:
333:
334: while ( index_rx != index_tx )
335: {
336:     error_code = SUCCESS;
337:     if ( dataset->quit_application ) error_code = -EC_APP_TERMINATE_CTRLC;
338:     if ( Character_Get( NULL ) ) error_code = -EC_APP_TERMINATE;
339:     if ( SUCCESS != error_code )
340:     {
341:         printf( "SPI_Data_Write_Read status ending receive loop\n" );
342:         goto SPI_Data_Write_Error_Exit;
343:     }
344:     SPI_Data_Write_Read_Helper_Read( board_id, object_size, tx_object_count, active_rx,
345:     &index_rx, rx_buffer );
346: }
347: return SUCCESS;
348: SPI_Data_Write_Error_Exit:
349:
350:
351: #if defined( SPI_PRINT_XFR_DEBUG )
352: printf( "SPI_Data_Write_Read:\n" );
353: #if defined( __MSDOS__ )
354: printf( " object_size      = %d\n", object_size );
355: printf( " tx_object_count  = %d\n", tx_object_count );
356: printf( " rx_object_count  = %d\n", rx_object_count );
357: printf( " index_tx        = %d\n", index_tx );
358: printf( " index_rx        = %d\n", index_rx );
359: printf( " tx_buffer        = %p\n", tx_buffer );

```

```

360: printf( " rx_buffer           = %p\n", rx_buffer );
361: #else
362: printf( " object_size          = %lu\n", object_size );
363: printf( " tx_object_count       = %lu\n", tx_object_count );
364: printf( " rx_object_count       = %lu\n", rx_object_count );
365: printf( " index_tx              = %lu\n", index_tx );
366: printf( " index_rx              = %lu\n", index_rx );
367: printf( " tx_buffer             = %p\n", tx_buffer );
368: printf( " rx_buffer             = %p\n", rx_buffer );
369: #endif
370: printf( " active_tx             = %s\n", active_tx ? "true" : "false" );
371: printf( " active_rx             = %s\n", active_rx ? "true" : "false" );
372:
373: printf( " global_internal_tx_byte_count = %u\n", global_internal_tx_byte_count );
374: printf( " global_internal_rx_byte_count = %u\n", global_internal_rx_byte_count );
375: #endif
376:
377:
378: #define FUNCTION_NAME__ "SPI_Data_Write_Read"
379: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
380: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, error_code );
381: #endif
382: #undef FUNCTION_NAME__
383:
384: return error_code;
385: }
```

3.2.1.258 SPI_Data_Write_Read_Helper_Commit Function

C++

```
static void SPI_Data_Write_Read_Helper_Commit(int board_id, size_t object_size, SPI_CSB_ENUM csb, BOOL *commit_ready, BOOL *commit_valid);
```

File

idi.c (see page 373)

Description

brief

returns true if user wishes to break.

Body Source

```

1: static void SPI_Data_Write_Read_Helper_Commit( int      board_id,
2:                                              size_t   object_size,
3:                                              SPI_CSB_ENUM csb,
4:                                              BOOL     commit_ready,
5:                                              BOOL *   commit_valid
6: )
7: {
8:
9:
10: (void) object_size;
11:
12:
13: #define FUNCTION_NAME__ "SPI_Data_Write_Read_Helper_Commit"
14: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
15: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
```

```

16: #endif
17: #undef FUNCTION_NAME__
18:
19:
20:
21: if ( true == commit_ready )
22: {
23:     if ( false == *commit_valid )
24:     {
25:         if ( CSB_SOFTWARE != csb ) SPI_Commit( board_id, 0xFF );
26:     }
27: #if defined( SPI_PRINT_DEBUG )
28: printf( "SPI_Data_Write_Helper_Commit: commit_valid = true\n" );
29: #endif
30:     *commit_valid = true;
31: }
32: else
33: {
34:     if ( CSB_BUFFER == csb )
35:     {
36:         if ( SPI_Commit_Is_Inactive( board_id ) )
37:         {
38: #if defined( SPI_PRINT_DEBUG )
39: printf( "SPI_Data_Write_Helper_Commit: commit_valid = false (CSB = BUFFER)\n" );
40: #endif
41:             *commit_valid = false;
42:         }
43:     }
44: else
45: {
46: #if defined( SPI_PRINT_DEBUG )
47: printf( "SPI_Data_Write_Helper_Commit: commit_valid = false\n" );
48: #endif
49:     *commit_valid = false;
50: }
51: }
52: }
```

3.2.1.259 SPI_Data_Write_Read_Helper_Read Function

C++

```
static void SPI_Data_Write_Read_Helper_Read(int board_id, size_t object_size, size_t
object_total_count, BOOL active_rx, size_t * index_rx, const void * rx_buffer);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static void SPI_Data_Write_Read_Helper_Read( int      board_id,
2:                                         size_t    object_size,
3:                                         size_t    object_total_count,
4:                                         BOOL     active_rx,
5:                                         size_t   * index_rx,
6:                                         const void * rx_buffer
```

```
7:             )
8: {
9:     BOOL rx_empty;
10:    int reg_symbol;
11:    size_t rx_bytes_available;
12:    size_t object_qty;
13:    size_t object_remaining;
14:    size_t index_object;
15:    size_t index;
16:    size_t index_byte;
17:    uint8_t bit_bucket;
18:
19: #if defined( SPI_PRINT_XFR_DEBUG )
20:     printf( " SPI_Data_Write_Read_Helper_Read: " );
21: #endif
22:
23: #define FUNCTION_NAME__ "SPI_Data_Write_Read_Helper_Read"
24: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
25:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
26: #endif
27: #undef FUNCTION_NAME__
28:
29:
30: SPI_Status_Read_FIFO_Status( board_id, &rx_empty, &rx_bytes_available );
31: if ( rx_bytes_available >= object_size )
32: {
33:     object_qty = rx_bytes_available / object_size;
34:     object_remaining = object_total_count - *index_rx;
35:     if ( object_qty > object_remaining )
36:     {
37:         object_qty = object_total_count - *index_rx;
38:     }
39:
40:     switch ( board_id )
41:     {
42:         case ID_IDI48: reg_symbol = (int) IDI_SPI_DATA; break;
43:         case ID_IDO48: reg_symbol = (int) IDO_SPI_DATA; break;
44:     }
45:
46:     if ( active_rx )
47:     {
48:         for ( index_object = 0; index_object < object_qty; index_object++ )
49:         {
50:             index_byte = (*index_rx) * object_size;
51:             for ( index = 0; index < object_size; index++ )
52:             {
53:                 IO_Read_U8( board_id, __LINE__, reg_symbol, &((uint8_t *) rx_buffer)[index_byte + index] );
54: #if defined( SPI_PRINT_XFR_DEBUG )
55:                 global_internal_rx_byte_count++;
56: #endif
57:             }
58:             *index_rx = *index_rx + 1;
59:         }
60:     }
61:     else
62:     {
63:         for ( index_object = 0; index_object < object_qty; index_object++ )
64:         {
65:             index_byte = (*index_rx) * object_size;
66:             for ( index = 0; index < object_size; index++ )
67:             {
68:                 IO_Read_U8( board_id, __LINE__, reg_symbol, &bit_bucket );
69: #if defined( SPI_PRINT_XFR_DEBUG )
```

```

70: global_internal_rx_byte_count++;
71: #endif
72:     }
73:     *index_rx = *index_rx + 1;
74:   }
75: }
76: #if defined( SPI_PRINT_XFR_DEBUG )
77: printf( "object_qty=%u, rx_bytes_available=%u, object_remaining=%u, object_size=%u",
78: object_qty, rx_bytes_available, object_remaining, object_size );
79: #endif
80: }
81: #if defined( SPI_PRINT_XFR_DEBUG )
82: printf( ", index_rx=%u\n", *index_rx );
83: #endif
84: }
```

3.2.1.260 SPI_Data_Write_Read_Helper_Write Function

C++

```
static void SPI_Data_Write_Read_Helper_Write(int board_id, size_t object_size, size_t
object_total_count, SPI_CSB_ENUM csb, BOOL active_tx, BOOL commit_valid, BOOL * commit_ready,
size_t * index_tx, const void * tx_buffer);
```

File

idi.c (see page 373)

Description

brief

Body Source

```

1: static void SPI_Data_Write_Read_Helper_Write( int      board_id,
2:                                         size_t    object_size,
3:                                         size_t    object_total_count,
4:                                         SPI_CSB_ENUM csb,
5:                                         BOOL      active_tx,
6:                                         BOOL      commit_valid,
7:                                         BOOL *    commit_ready,
8:                                         size_t *  index_tx,
9:                                         const void * tx_buffer
10:                                        )
11: {
12:   BOOL      tx_full;
13:   BOOL      tx_empty;
14:   int       reg_symbol;
15:   size_t    tx_bytes_available;
16:   size_t    object_qty;
17:   size_t    object_remaining;
18:   size_t    index_object;
19:   size_t    index;
20:   size_t    index_byte;
21:
22: #if defined( SPI_PRINT_XFR_DEBUG )
23:   int      reg_tx_status, reg_rx_status;
24:   uint8_t  reg_tx_status_value, reg_rx_status_value;
25: #endif
26:
```

```
27: const uint8_t    tx_dummy_value = 0x00;
28:
29:
30: #if defined( SPI_PRINT_XFR_DEBUG )
31: printf( " SPI_Data_Write_Read_Helper_Write: " );
32: #endif
33:
34: #define FUNCTION_NAME__ "SPI_Data_Write_Read_Helper_Write"
35: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
36: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
37: #endif
38: #undef FUNCTION_NAME__
39:
40: if ( false == commit_valid )
41: {
42: #if defined( SPI_PRINT_XFR_DEBUG )
43: printf( " GO" );
44: #endif
45: if ( false == *commit_ready )
46: {
47: #if defined( SPI_PRINT_XFR_DEBUG )
48: printf( ", FILL" );
49: #endif
50:
51: SPI_Status_Write_FIFO_Status( board_id, &tx_full, &tx_empty, &tx_bytes_available );
52: if ( tx_bytes_available >= object_size )
53: {
54: object_qty = tx_bytes_available / object_size;
55: object_remaining = object_total_count - *index_tx;
56: if ( object_qty > object_remaining )
57: {
58: object_qty = object_total_count - *index_tx;
59: }
60:
61: if ( CSB_BUFFER == csb )
62: {
63: if ( object_qty >= 1 ) object_qty = 1;
64: }
65: #if defined( SPI_PRINT_DEBUG )
66: printf( " SPI_Data_Write_Read_Helper_Write: TX object_qty = %d\n", (int) object_qty );
67: #endif
68: switch ( board_id )
69: {
70: case ID_IDI48:
71: reg_symbol = (int) IDI_SPI_DATA;
72: #if defined( SPI_PRINT_XFR_DEBUG )
73: reg_tx_status = IDI_SPI_TX_STATUS;
74: reg_rx_status = IDI_SPI_RX_STATUS;
75: #endif
76: break;
77: case ID_IDO48:
78: reg_symbol = (int) IDO_SPI_DATA;
79: #if defined( SPI_PRINT_XFR_DEBUG )
80: reg_tx_status = IDO_SPI_TX_STATUS;
81: reg_rx_status = IDO_SPI_RX_STATUS;
82: #endif
83: break;
84: }
85:
86: if ( active_tx )
87: {
88: #if defined( SPI_PRINT_DEBUG )
89: printf( " SPI_Data_Write_Read_Helper_Write: TX active\n" );
90: #endif
```

```

91:
92: #if defined( SPI_PRINT_XFR_DEBUG )
93: printf( " , DSTATUS:" );
94: #endif
95:     for ( index_object = 0; index_object < object_qty; index_object++ )
96:     {
97: #if defined( SPI_PRINT_DEBUG )
98: printf( "   SPI_Data_Write_Read_Helper_Write: TX bytes:" );
99: #endif
100:     index_byte = *index_tx * object_size;
101:     for ( index = 0; index < object_size; index++ )
102:     {
103: #if defined( SPI_PRINT_DEBUG )
104: printf( " 0x%02X", ((uint8_t *) tx_buffer)[index_byte + index] );
105: #endif
106:     IO_Write_U8( board_id, __LINE__, reg_symbol, ((uint8_t *) tx_buffer)[index_byte +
index] );
107:
108:
109: #if defined( SPI_PRINT_XFR_DEBUG )
110: global_internal_tx_byte_count++;
111: # if defined( __MSDOS__ )
112: delay( 200 );
113: #endif
114: IO_Read_U8( board_id, __LINE__, reg_tx_status, &reg_tx_status_value );
115: IO_Read_U8( board_id, __LINE__, reg_rx_status, &reg_rx_status_value );
116: printf( " [0x%02X.0x%02X]", reg_tx_status_value, reg_rx_status_value );
117: #endif
118:
119:
120:     }
121:     *index_tx = *index_tx + 1;
122: }
123: #if defined( SPI_PRINT_DEBUG )
124: printf( "\n" );
125: #endif
126: }
127: else
128: {
129: #if defined( SPI_PRINT_DEBUG )
130: printf( "   SPI_Data_Write_Read_Helper_Write: TX inactive\n" );
131: #endif
132:
133:
134: #if defined( SPI_PRINT_XFR_DEBUG )
135: printf( " , NSTATUS:" );
136: #endif
137:     for ( index_object = 0; index_object < object_qty; index_object++ )
138:     {
139:
140:         for ( index = 0; index < object_size; index++ )
141:         {
142:             IO_Write_U8( board_id, __LINE__, reg_symbol, tx_dummy_value );
143:
144: #if defined( SPI_PRINT_XFR_DEBUG )
145: global_internal_tx_byte_count++;
146: # if defined( __MSDOS__ )
147: delay( 200 );
148: #endif
149: IO_Read_U8( board_id, __LINE__, reg_tx_status, &reg_tx_status_value );
150: IO_Read_U8( board_id, __LINE__, reg_rx_status, &reg_rx_status_value );
151: printf( " [0x%02X.0x%02X]", reg_tx_status_value, reg_rx_status_value );
152: #endif
153:
```

```
154:         }
155:         *index_tx = *index_tx + 1;
156:     }
157:
158:     }
159:     *commit_ready = true;
160: #if defined( SPI_PRINT_XFR_DEBUG )
161: printf( " , object_qty=%u, tx_bytes_available=%u, object_remaining=%u, object_size=%u",
object_qty, tx_bytes_available, object_remaining, object_size );
162: printf( " , index_tx=%u", *index_tx );
163: #endif
164:
165: #if defined( SPI_PRINT_DEBUG )
166: printf( "SPI_Data_Write_Read_Helper_Write TX request\n" );
167: #endif
168:     }
169:     else
170:     {
171:
172: printf( " , FULL" );
173:     }
174:     }
175:     else
176:     {
177:
178: printf( " , FULL" );
179:     }
180:     }
181:     else
182:     {
183: #if defined( SPI_PRINT_DEBUG )
184: printf( "SPI_Data_Write_Read_Helper_Write TX ack\n" );
185: #endif
186: #if defined( SPI_PRINT_XFR_DEBUG )
187: printf( " ACK" );
188: #endif
189:     *commit_ready = false;
190:     }
191: #if defined( SPI_PRINT_XFR_DEBUG )
192: printf( "\n" );
193: #endif
194: }
```

3.2.1.261 SPI_FIFO_Read Function

C++

```
int SPI_FIFO_Read(int board_id, const void * buffer, size_t size, size_t count, FILE * fd_log);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned indicating an error. In addition, a positive value is returned indicating the number of actual objects written.

Description

brief Reads from the SPI receive/read data FIFO. It does not attempt to correlate the number of transmit bytes with receive bytes. Its purpose is more for low level hardware testing. Note that this function has a signature identical to the fread() function (i.e. make use of function pointers to guide sourcing of data).

param[in] buffer (see page 325) Buffer for the data destination. **param[in] size** Size of objects in bytes. **param[in] count** Number of objects to be read **param[out] fd_log** Optional log file to write the buffer (see page 325) too. If NULL, then no logging.

Body Source

```

1: int SPI_FIFO_Read( int board_id, const void * buffer, size_t size, size_t count, FILE * fd_log )
2: {
3:     int error_code;
4:     size_t bytes_available;
5:     BOOL empty;
6:     size_t index;
7:     size_t qty_objects;
8:     size_t qty_bytes;
9:     int reg_symbol;
10:
11: #define FUNCTION_NAME__ "SPI_FIFO_Read"
12: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
13:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
14: #endif
15: #undef FUNCTION_NAME__
16:
17:     error_code = SUCCESS;
18:
19:     if ( (size * count) > SPI_FIFO_SIZE ) return -EC_PARAMETER;
20:
21:     if ( SPI_IsNotPresent( board_id ) ) return -EC_SPI_NOT_FOUND;
22:
23:     SPI_Status_Read_FIFO_Status( board_id, &empty, &bytes_available );
24:
25:     qty_objects = bytes_available / size;
26:     if ( count < qty_objects ) qty_objects = count;
27:
28:     qty_bytes = qty_objects * size;
29:
30:     switch ( board_id )
31:     {
32:         case ID_IDI48: reg_symbol = (int) IDI_SPI_DATA; break;
33:         case ID_IDO48: reg_symbol = (int) IDO_SPI_DATA; break;
34:     }
35:     for ( index = 0; index < qty_bytes; index++ ) IO_Read_U8( board_id, __LINE__, reg_symbol,
&((uint8_t *) buffer)[index] );
36:
37:     if ( NULL != fd_log )
38:     {
39:         error_code = fwrite( buffer, size, qty_objects, fd_log );
40:     }
41:
42:     if ( SUCCESS == error_code ) error_code = ( (int) qty_objects );
43:     return error_code;
44: }
```

3.2.1.262 SPI_FIFO_Write Function

C++

```
int SPI_FIFO_Write(int board_id, const void * buffer, size_t size, size_t count, FILE * fd_log);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned indicating an error. In addition, a positive value is returned indicating the number of actual objects written.

Description

brief Writes specifically to the SPI transmit/write data FIFO. It does not attempt to correlate the number of transmit bytes with receive bytes. Its purpose is more for low level hardware testing. Note that this function has a signature identical to the fwrite() function (i.e. make use of function pointers to guide destination of data).

param[in] buffer (see page 325) Buffer containing the data to be written. param[in] size Size of objects in bytes. param[in] count Number of objects to be written param[out] fd_log Optional log file to write the buffer (see page 325) too. If NULL, then no logging.

Body Source

```
1: int SPI_FIFO_Write( int board_id, const void * buffer, size_t size, size_t count, FILE * fd_log )
2: {
3:     int error_code;
4:     size_t bytes_available;
5:     BOOL empty;
6:     BOOL full;
7:     size_t index;
8:     size_t qty_objects;
9:     size_t qty_bytes;
10:    int reg_symbol;
11:
12: #define FUNCTION_NAME__ "SPI_FIFO_Write"
13: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
14:    IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
15: #endif
16: #undef FUNCTION_NAME__
17:
18:    error_code = SUCCESS;
19:
20:    if ( (size * count) > SPI_FIFO_SIZE ) return -EC_PARAMETER;
21:
22:    if ( SPI_IsNotPresent( board_id ) ) return -EC_SPI_NOT_FOUND;
23:
24:    SPI_Status_Write_FIFO_Status( board_id, &full, &empty, &bytes_available );
25:
26:    qty_objects = bytes_available / size;
27:    if ( count < qty_objects ) qty_objects = count;
28:
29:    qty_bytes = qty_objects * size;
```

```

30:
31:
32:
33:
34:     switch ( board_id )
35:     {
36:         case ID_IDI48: reg_symbol = (int) IDI_SPI_DATA; break;
37:         case ID_IDO48: reg_symbol = (int) IDO_SPI_DATA; break;
38:     }
39:     for ( index = 0; index < qty_bytes; index++ ) IO_Write_U8( board_id, __LINE__,
reg_symbol, ((uint8_t *) buffer)[index] );
40:
41:     if ( NULL != fd_log )
42:     {
43:         error_code = fwrite( buffer, size, qty_objects, fd_log );
44:     }
45:
46:     if ( SUCCESS == error_code ) error_code = ( (int) qty_objects );
47:     return error_code;
48: }
```

3.2.1.263 SPI_ID_Get Function

C++

```
int SPI_ID_Get(int board_id, uint16_t * id);
```

File

idi.c (see page 373)

Description

This is function SPI_ID_Get.

Body Source

```

1: int SPI_ID_Get( int board_id, uint16_t * id )
2: {
3:     #define FUNCTION_NAME__ "SPI_ID_Get"
4:     #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
5:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
6:     #endif
7:     #undef FUNCTION_NAME__
8:
9:     switch ( board_id )
10:    {
11:        case ID_IDI48:
12:            return SPI_ID_Get_Helper( board_id, IDI_SPI_ID_LSB, IDI_SPI_ID_MSB, id );
13:        case ID_IDO48:
14:            return SPI_ID_Get_Helper( board_id, IDO_SPI_ID_LSB, IDO_SPI_ID_MSB, id );
15:    }
16:    return -EC_BOARD;
17: }
```

3.2.1.264 SPI_ID_Get_Helper Function

C++

```
static int SPI_ID_Get_Helper(int board_id, int id_lsb, int id_msb, uint16_t * id);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Retrieves the SPI ID register value.

param[out] id The SPI component ID value.

Body Source

```
1: static int SPI_ID_Get_Helper( int board_id, int id_lsb, int id_msb, uint16_t * id )
2: {
3:     uint8_t     lsb, msb;
4:     struct board_dataset * dataset;
5:
6:     dataset = (struct board_dataset *) board_definition[board_id].dataset;
7:
8: #define FUNCTION_NAME__ "SPI_ID_Get_Helper"
9: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
10:    IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
11: #endif
12: #undef FUNCTION_NAME__
13:
14: if ( MODE_JUMPERS_0 == dataset->mode_jumpers )
15: {
16:     return -EC_MODE_LEGACY;
17: }
18:
19: IO_Read_U8( board_id, __LINE__, id_lsb, &lsb );
20: IO_Read_U8( board_id, __LINE__, id_msb, &msb );
21: *id = ( ((uint16_t) msb) << 8 ) | ((uint16_t) lsb);
22: #if defined( ID_ALWAYS_REPORT_AS_GOOD )
23: *id = ID_SPI;
24: #endif
25: return SUCCESS;
26: }
```

3.2.1.265 SPI_IsNotPresent Function

C++

```
int SPI_IsNotPresent(int board_id);
```

File

idi.c (see page 373)

Returns

A zero is returned if the SPI component ID is not found within the register space.

Description

brief Reports if the SPI component is available within the register space by matching a known ID. The SPI register map is only enabled within the hardware if the hardware mode is not zero (i.e. M1 and M0 jumpers on the board provide a nonzero value).

Body Source

```

1: int SPI_IsNotPresent( int board_id )
2: {
3:     int         error_code;
4:     uint16_t    id;
5:     struct board_dataset * dataset;
6:     dataset = (struct board_dataset *) board_definition[board_id].dataset;
7:
8: #define FUNCTION_NAME__ "SPI_IsNotPresent"
9: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
10: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
11: #endif
12: #undef FUNCTION_NAME__
13:
14:
15:     if ( true == dataset->spi_is_present ) return 0;
16:
17:     error_code = SPI_ID_Get( board_id, &id );
18:     if ( error_code < 0 ) return error_code;
19:     if ( ID_SPI == id )
20:     {
21:         dataset->spi_is_present = true;
22:     }
23:     return 0;
24: }
25: return 1;
26: }
```

3.2.1.266 SPI_Report_Configuration_Text Function

C++

```
int SPI_Report_Configuration_Text(struct spi_cfg * cfg, FILE * out);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Creates a human readable report of the SPI configuration data structure.

param[in] cfg SPI configuration data structure pointer param[out] out File destination descriptor

Body Source

```

1: int SPI_Report_Configuration_Text( struct spi_cfg * cfg, FILE * out )
2: {
3:     fprintf( out, "##### SPI Configuration:\n" );
4:     fprintf( out, "sdio_wrap           = %s\n", cfg->sdio_wrap ? "true" : "false" );
5:     fprintf( out, "sdo_polarity         = %s\n", cfg->sdo_polarity ? "true" : "false" );
6:     fprintf( out, "sdi_polarity         = %s\n", cfg->sdi_polarity ? "true" : "false" );
7:     fprintf( out, "sclk_phase          = %s\n", cfg->sclk_phase ? "true" : "false" );
8:     fprintf( out, "sclk_polarity        = %s\n", cfg->sclk_polarity ? "true" : "false" );
9:     fprintf( out, "chip_select_route_ch1 = %s\n", cfg->chip_select_route_ch1 ? "true" :
"false" );
10:    fprintf( out, "chip_select_behavior = " );
11:
12:    switch( cfg->chip_select_behavior )
13:    {
14:        case CSB_SOFTWARE: fprintf( out, "CSB_SOFTWARE" ); break;
15:        case CSB_BUFFER:   fprintf( out, "CSB_BUFFER" ); break;
16:        case CSB_uint8_t:  fprintf( out, "CSB_uint8_t" ); break;
17:        case CSB_uint16_t: fprintf( out, "CSB_uint16_t" ); break;
18:        default:           fprintf( out, "undefined" ); break;
19:    }
20:    fprintf( out, "\n" );
21:
22:    fprintf( out, "end_cycle_delay      = 0x%02X (%d)\n", cfg->end_cycle_delay,
cfg->end_cycle_delay );
23:    fprintf( out, "half_clock_interval  = 0x%04X (%d)\n", cfg->half_clock_interval,
cfg->half_clock_interval );
24:
25:    fprintf( out, "clock_hz              = %f Hz\n", cfg->clock_hz );
26:    fprintf( out, "end_delay_ns          = %f ns\n", cfg->end_delay_ns );
27:    fprintf( out, "\n" );
28:    return SUCCESS;
29: }
```

3.2.1.267 SPI_Report_Status_Text Function

C++

```
int SPI_Report_Status_Text(struct spi_status * status, FILE * out);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Produces a human readable report of the SPI status data structure.

param[in] status SPI status data structure pointer param[out] out File destination descriptor

Body Source

```

1: int SPI_Report_Status_Text( struct spi_status * status, FILE * out )
```

```

2: {
3:   fprintf( out, "######################################## SPI " );
4:   if ( status->tx_status ) fprintf( out, "TX" );
5:   else                      fprintf( out, "RX" );
6:
7:   fprintf( out, " Status:\n" );
8:
9:   fprintf( out, "full      = %s\n", status->full ? "true" : "false" );
10:  fprintf( out, "empty     = %s\n", status->empty ? "true" : "false" );
11:  fprintf( out, "fifo size = %d\n", status->fifo_size );
12:  fprintf( out, "fifo count = %d\n", status->fifo_count );
13:
14:  return SUCCESS;
15: }
```

3.2.1.268 SPI_Status_Read Function

C++

```
int SPI_Status_Read(int board_id, struct spi_status * status);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Builds a detailed status data structure of the receive/read incoming SPI data FIFO. Reports the quantity of bytes currently in the receive FIFO, full flag, empty flag, the total size of the FIFO in bytes, and sets tx_status to false indicating that this is status specific to the receive FIFO.

The status register has the following format: status[7] full status[6] empty status[5] not used (future size expansion) status[4:0] number of bytes currently in the FIFO

param[out] status Pointer to status data structure to be updated.

Body Source

```

1: int SPI_Status_Read( int board_id, struct spi_status * status )
2: {
3:   uint8_t reg_rx_status;
4:
5: #define FUNCTION_NAME__ "SPI_Status_Read"
6: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
7:   IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
8: #endif
9: #undef FUNCTION_NAME__
10:
11: if ( SPI_IsNotPresent( board_id ) ) return -EC_SPI_NOT_FOUND;
12:
13: switch ( board_id )
14: {
15:   case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_RX_STATUS, &reg_rx_status );
16:   break;
16:   case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_RX_STATUS, &reg_rx_status );
```

```

break;
17: }
18:
19: status->fifo_count = (int)( reg_rx_status & 0x1F );
20: status->full      = (BOOL)( reg_rx_status & 0x80 );
21: status->fifo_size = (int) SPI_FIFO_SIZE;
22: status->empty     = (BOOL)( reg_rx_status & 0x40 );
23: status->tx_status = false;
24: return SUCCESS;
25: }

```

3.2.1.269 SPI_Status_Read_FIFO_Is_Not_Empty Function

C++

```
BOOL SPI_Status_Read_FIFO_Is_Not_Empty(int board_id);
```

File

idi.c (see page 373)

Returns

Returns true if the receive/read FIFO is not empty.

Description

brief Returns the receive/read FIFO empty status flag. This function is typically used to determine if the FIFO is empty.

Body Source

```

1: BOOL SPI_Status_Read_FIFO_Is_Not_Empty( int board_id )
2: {
3:     uint8_t reg_rx_status;
4:
5: #define FUNCTION_NAME__ "SPI_Status_Read_FIFO_Is_Not_Empty"
6: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
7:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
8: #endif
9: #undef FUNCTION_NAME__
10:
11: switch ( board_id )
12: {
13:     case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_RX_STATUS, &reg_rx_status );
break;
14:     case IDIDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_RX_STATUS, &reg_rx_status );
break;
15: }
16: if ( reg_rx_status & 0x40 ) return false;
17: return true;
18: }

```

3.2.1.270 SPI_Status_Read_FIFO_Status Function

C++

```
void SPI_Status_Read_FIFO_Status(int board_id, BOOL * empty, size_t * bytes_available);
```

File

idi.c (see page 373)

Returns

nothing

Description

brief Returns the complete read/receive FIFO status.

The status register has the following format: status[7] full status[6] empty status[5] not used (future size expansion) status[4:0] number of bytes currently in the FIFO

param[out] empty FIFO empty flag param[out] bytes_available a count of the number of bytes in the FIFO

Body Source

```

1: void SPI_Status_Read_FIFO_Status( int board_id, BOOL * empty, size_t * bytes_available )
2: {
3: #if defined( SPI_PRINT_XFR_DEBUG )
4:     uint8_t reg_value[4];
5:     size_t index;
6:     size_t scratch;
7:     int error_code;
8: #endif
9:     uint8_t reg_rx_status;
10:
11: #define FUNCTION_NAME__ "SPI_Status_Read_FIFO_Status"
12: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
13:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
14: #endif
15: #undef FUNCTION_NAME__
16:
17:
18: #if defined( SPI_PRINT_XFR_DEBUG )
19:     switch ( board_id )
20:     {
21:         case ID_IDI48:
22:             for ( index = 0; index < 4; index++ )
23:                 IO_Read_U8( board_id, __LINE__, IDI_SPI_RX_STATUS, &(reg_value[index]) );
24:             break;
25:         case ID_IDO48:
26:             for ( index = 0; index < 4; index++ )
27:                 IO_Read_U8( board_id, __LINE__, IDO_SPI_RX_STATUS, &(reg_value[index]) );
28:             break;
29:     }
30:
31:     error_code = 0;
32:     for ( index = 1; index < 4; index++ )
33:     {
34:         scratch = (size_t)( reg_value[index] & SPI_FIFO_SIZE_BIT_WIDTH );
35:         if ( scratch < ( (size_t)( reg_value[0] & SPI_FIFO_SIZE_BIT_WIDTH ) ) )
36:         {
37:             error_code = -1;
38:         }
39:
40:         scratch = (size_t) ( reg_value[index] & 0x40 );
41:         if ( scratch != ( (size_t) ( reg_value[0] & 0x40 ) ) )
42:     }
```

```

43:     error_code = -1;
44: }
45: }
46:
47: if ( error_code < 0 )
48: {
49:     printf( "###ERROR: SPI_Status_Read_FIFO_Status: " );
50:     for ( index = 0; index < 4; index++ ) printf( " 0x%02X", reg_value[index] );
51:     printf( "\n" );
52: }
53: reg_rx_status = reg_value[0];
54: #else
55: switch ( board_id )
56: {
57:     case ID_IDI48:
58:         IO_Read_U8( board_id, __LINE__, IDI_SPI_RX_STATUS, &reg_rx_status );
59:         break;
60:     case ID_IDO48:
61:         IO_Read_U8( board_id, __LINE__, IDO_SPI_RX_STATUS, &reg_rx_status );
62:         break;
63: }
64: #endif
65:
66:
67: *bytes_available = (size_t)( reg_rx_status & SPI_FIFO_SIZE_BIT_WIDTH );
68: *empty           = (BOOL)( reg_rx_status & 0x40 );
69: }
```

3.2.1.271 SPI_Status_Write Function

C++

```
int SPI_Status_Write(int board_id, struct spi_status * status);
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief Builds a detailed status data structure of the transmit/write outgoing SPI data FIFO. Reports the quantity of bytes currently in the transmit FIFO, full flag, empty flag, the total size of the FIFO in bytes, and sets tx_status to true indicating that this is status specific to the transmit FIFO.

The status register has the following format: status[7] full status[6] empty status[5] not used (future size expansion) status[4:0] number of bytes currently in the FIFO

param[out] status Pointer to status data structure to be updated.

Body Source

```

1: int SPI_Status_Write( int board_id, struct spi_status * status )
2: {
3:     uint8_t reg_tx_status;
4:
```

```

5: #define FUNCTION_NAME__ "SPI_Status_Write"
6: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
7: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
8: #endif
9: #undef FUNCTION_NAME__
10:
11: if ( SPI_IsNotPresent( board_id ) ) return -EC_SPI_NOT_FOUND;
12:
13: switch ( board_id )
14: {
15:   case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_TX_STATUS, &reg_tx_status );
16:   break;
17:   case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_TX_STATUS, &reg_tx_status );
18:   break;
19: }
20: status->fifo_count = (int)( reg_tx_status & SPI_FIFO_SIZE_BIT_WIDTH );
21: status->full      = (BOOL)( reg_tx_status & 0x80 );
22: status->fifo_size = (int) SPI_FIFO_SIZE;
23: status->empty     = (BOOL)( reg_tx_status & 0x40 );
24: status->tx_status = true;
25: return SUCCESS;
}

```

3.2.1.272 SPI_Status_Write_FIFO_Is_Full Function

C++

```
BOOL SPI_Status_Write_FIFO_Is_Full(int board_id);
```

File

idi.c (see page 373)

Returns

Returns true if the transmit/write FIFO is full.

Description

brief Returns the transmit/write FIFO full status flag. It is preferable to use the SPI_Status_Write (see page 286)() or SPI_Status_Write_FIFO_Status (see page 288)() because all status is retrieved at one time.

Body Source

```

1: BOOL SPI_Status_Write_FIFO_Is_Full( int board_id )
2: {
3:   uint8_t reg_tx_status;
4:
5: #define FUNCTION_NAME__ "SPI_Status_Write_FIFO_Is_Full"
6: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
7: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
8: #endif
9: #undef FUNCTION_NAME__
10:
11: switch ( board_id )
12: {
13:   case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_TX_STATUS, &reg_tx_status );
14:   break;
15:   case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_TX_STATUS, &reg_tx_status );
16:   break;
17: }
18: status->fifo_count = (int)( reg_tx_status & SPI_FIFO_SIZE_BIT_WIDTH );
19: status->full      = (BOOL)( reg_tx_status & 0x80 );
20: status->fifo_size = (int) SPI_FIFO_SIZE;
21: status->empty     = (BOOL)( reg_tx_status & 0x40 );
22: status->tx_status = true;
23: return SUCCESS;
}

```

```

break;
15: }
16: if ( reg_tx_status & 0x80 ) return true;
17: return false;
18: }

```

3.2.1.273 SPI_Status_Write_FIFO_Is_Not_Empty Function

C++

```
BOOL SPI_Status_Write_FIFO_Is_Not_Empty(int board_id);
```

File

idi.c (see page 373)

Returns

Returns true if the transmit/write FIFO is not empty.

Description

brief Returns the transmit/write FIFO empty status flag. This function is typically used to wait for the transmit/write FIFO to become empty.

Body Source

```

1: BOOL SPI_Status_Write_FIFO_Is_Not_Empty( int board_id )
2: {
3:     uint8_t reg_tx_status;
4:
5: #define FUNCTION_NAME__ "SPI_Status_Write_FIFO_Is_Not_Empty"
6: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
7:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
8: #endif
9: #undef FUNCTION_NAME__
10:
11:    switch ( board_id )
12:    {
13:        case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_TX_STATUS, &reg_tx_status );
break;
14:        case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_TX_STATUS, &reg_tx_status );
break;
15:    }
16:    if ( reg_tx_status & 0x40 ) return false;
17:    return true;
18: }

```

3.2.1.274 SPI_Status_Write_FIFO_Status Function

C++

```
void SPI_Status_Write_FIFO_Status(int board_id, BOOL * full, BOOL * empty, size_t * bytes_available);
```

File

idi.c (see page 373)

Returns

nothing

Description

brief Returns the complete write/transmit FIFO status.

The status register has the following format: status[7] full status[6] empty status[5] not used (future size expansion) status[4:0] number of bytes currently in the FIFO

param[out] full FIFO full flag param[out] empty FIFO empty flag param[out] bytes_in_fifo a count of the number of bytes in the FIFO

Body Source

```

1: void SPI_Status_Write_FIFO_Status( int board_id, BOOL * full, BOOL * empty, size_t * bytes_available )
2: {
3: #if defined( SPI_PRINT_XFR_DEBUG )
4: uint8_t reg_buf[4];
5: size_t scratch;
6: size_t index;
7: int error_code;
8: #endif
9: uint8_t reg_tx_status;
10:
11: #define FUNCTION_NAME__ "SPI_Status_Write_FIFO_Status"
12: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
13: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
14: #endif
15: #undef FUNCTION_NAME__
16:
17:
18: #if defined( SPI_PRINT_XFR_DEBUG )
19: switch ( board_id )
20: {
21: case ID_IDI48:
22: for ( index = 0; index < 4; index++ )
23: IO_Read_U8( board_id, __LINE__, IDI_SPI_TX_STATUS, &(reg_buf[index]) );
24: break;
25: case ID_IDO48:
26: for ( index = 0; index < 4; index++ )
27: IO_Read_U8( board_id, __LINE__, IDO_SPI_TX_STATUS, &(reg_buf[index]) );
28: break;
29: }
30:
31: error_code = 0;
32: for ( index = 1; index < 4; index++ )
33: {
34: scratch = (size_t)( reg_buf[index] & SPI_FIFO_SIZE_BIT_WIDTH );
35: if ( scratch > ( (size_t)( reg_buf[0] & SPI_FIFO_SIZE_BIT_WIDTH ) ) )
36: {
37: error_code = -1;
38: }
39: scratch = (size_t) ( reg_buf[index] & 0x40 );

```

```

41:   if ( scratch != ( (size_t) ( reg_buf[0] & 0x40 ) ) )
42:   {
43:     error_code = -1;
44:   }
45: }
46:
47: if ( error_code < 0 )
48: {
49:   printf( "###ERROR: SPI_Status_Write_FIFO_Status: " );
50:   for ( index = 0; index < 4; index++ ) printf( ", 0x%02X", reg_buf[index] );
51:   printf( "\n" );
52: }
53:
54: reg_tx_status = reg_buf[0];
55: #else
56: switch ( board_id )
57: {
58:   case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_TX_STATUS, &reg_tx_status );
break;
59:   case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_TX_STATUS, &reg_tx_status );
break;
60: }
61: #endif
62:
63: switch( reg_tx_status & 0xC0 )
64: {
65:   case 0x00: *full = false; *empty = false; break;
66:   case 0x40: *full = false; *empty = true; break;
67:   case 0x80: *full = true; *empty = false; break;
68:   case 0xC0:
69:     *full = true;
70:     *empty = true;
71: #define FUNCTION_NAME__ "SPI_Status_Write_FIFO_Status-FullEmpty"
72: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
73:   IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
74: #endif
75: #undef FUNCTION_NAME__
76:
77:   break;
78: }
79:
80: *bytes_available = (size_t) SPI_FIFO_SIZE_BIT_WIDTH - (size_t)( reg_tx_status &
SPI_FIFO_SIZE_BIT_WIDTH );
81: }
```

3.2.1.275 StopWatch_Close Function

C++

```
void StopWatch_Close(STOPWATCH_HANDLE_TYPE handle);
```

File

stopwatch.h ( see page 580)

Description

This is function StopWatch_Close.

3.2.1.276 StopWatch_Initialization Function

C++

```
void StopWatch_Initialization();
```

File

stopwatch.h (see page 580)

Description

This is function StopWatch_Initialization.

3.2.1.277 StopWatch_Open Function

C++

```
STOPWATCH_HANDLE_TYPE StopWatch_Open();
```

File

stopwatch.h (see page 580)

Description

This is function StopWatch_Open.

3.2.1.278 StopWatch_Process Function

C++

```
void StopWatch_Process();
```

File

stopwatch.h (see page 580)

Description

This is function StopWatch_Process.

3.2.1.279 StopWatch_Set Function

C++

```
void StopWatch_Set(STOPWATCH_HANDLE_TYPE handle, STOPWATCH_STATE_TYPE state);
```

File

stopwatch.h (see page 580)

Description

This is function StopWatch_Set.

3.2.1.280 StopWatch_Termination Function

C++

```
void StopWatch_Termination();
```

File

stopwatch.h (see page 580)

Description

This is function StopWatch_Termination.

3.2.1.281 StopWatch_TimeStamp_String Function

C++

```
void StopWatch_TimeStamp_String(char * return_string, int return_string_size);
```

File

stopwatch.h (see page 580)

Description

This is function StopWatch_TimeStamp_String.

3.2.1.282 StopWatch_Value Function

C++

```
STOPWATCH_TIMEVAL_TYPE * StopWatch_Value(STOPWATCH_HANDLE_TYPE handle);
```

File

stopwatch.h (see page 580)

Description

This is function StopWatch_Value.

3.2.1.283 String_To_Bool Function

C++

```
BOOL String_To_Bool(const char * str);
```

File

idi.c (see page 373)

Returns

a BOOL (see page 318) is returned. The default value returned is false.

Description

brief General function used to convert a string into a boolean equivalent value.

param[in] str string input for conversion.

Body Source

```
1: BOOL String_To_Bool( const char * str )
2: {
3:     switch( str[0] )
4:     {
5:         case '0':
6:         case 'f':
7:         case 'F':
8:             return false;
9:         case '1':
10:        case 't':
11:        case 'T':
12:            return true;
13:     }
14:     return false;
15: }
```

3.2.1.284 Termination Function

C++

```
int Termination(int board_id);
```

File

idi.c (see page 373)

Returns

SUCCESS (0) if no errors encountered, otherwise errors are reported as a negative value.

Description

brief Runs upon application exit. It saves the idi_dataset (see page 302) data structure.

Body Source

```

1: int Termination( int board_id )
2: {
3:     int error_code;
4:
5:
6:
7:
8:
9: #if defined( MAIN_DEBUG )
10: printf( "Main: Termination\n" );
11: #endif
12:
13: error_code = IO_Trace_Dump( board_id );
14: switch( board_id )
15: {
16:     case ID_IDI48:
17:         error_code = IDI_Termination( (struct idi_dataset *) board_definition[board_id].dataset
);
18:         break;
19:     case ID_IDO48:
20:         error_code = IDO_Termination( (struct ido_dataset *) board_definition[board_id].dataset
);
21:         break;
22: }
23: return error_code;
24: }
```

3.2.1.285 Time_Current_String Function

C++

```
int Time_Current_String(char * buf, size_t buf_size);
```

File

idi.c (see page 373)

Returns

SUCCESS always.

Description

brief Obtains the current time and date. This is exerpetted from AES Universal Library/Driver.

Excerpt from AES advanced Linux library/driver. COPYRIGHT NOTICE Copyright (c) 2012 by Apex Embedded Systems.

param[out] buf resulting date-time string. param[in] buf_size the character size of the buffer (see page 325) including null terminating character.

Body Source

```

1: int Time_Current_String( char * buf, size_t buf_size )
2: {
3:     struct tm * time_block;
4:     time_t time_current;
5:     char * time_ascii;
```

```

6: size_t index;
7:
8: time_current = time( NULL );
9:
10: time_block = localtime( &time_current );
11:
15: time_ascii = asctime( time_block );
16: index = 0;
17:
18: while( ( time_ascii[index] != '\n' ) && ( index < buf_size ) )
19: {
20:     buf[index] = time_ascii[index];
21:     index++;
22: }
23: if ( index >= buf_size ) index = buf_size - 1;
24: buf[index] = '\0';
25: return SUCCESS;
26: }
```

3.2.2 Structs, Records, Enums

The following table lists structs, records, enums in this documentation.

Enumerations

	Name	Description
☞	buffer_column (☞ see page 299)	brief
☞	CLM_BTS (☞ see page 299)	brief
☞	IO_TRACE_ENUM (☞ see page 306)	
☞	IO_TRACE_START_ENUM (☞ see page 307)	
☞	IO_TRACE_STOP_ENUM (☞ see page 307)	
☞	irqreturn (☞ see page 308)	<p>The following code is directly from irqreturn.h from the Linux kernel tree. This is being used by the simulator and DOS to emulate Linux driver behavior and specifically resource allocations.</p> <ul style="list-style-type: none"> * • enum irqreturn • @IRQ_NONE interrupt was not from this device • @IRQ_HANDLED interrupt was handled by this device • @IRQ_WAKE_THREAD handler requests to wake the handler thread
	BANK_ENUM (☞ see page 312)	This is type BANK_ENUM.
	EC_ENUM (☞ see page 313)	EC = Error Code
	IDI_BANK_ENUM (☞ see page 313)	This is type IDI_BANK_ENUM.
	IDI_REG_ENUM (☞ see page 313)	brief Once the register location (i.e. logical_address or row) is known, then all other parameters associated with that register can be looked up including physical_offset, bank and legal read/write access.
	IDO_BANK_ENUM (☞ see page 314)	This is type IDO_BANK_ENUM.
	IDO_REG_ENUM (☞ see page 314)	

	IOKERN_CHAIN_TO_OLD_ENUM (see page 314)	
	IOKERN_IRQ_ENUM (see page 315)	
	IOKERN_TASK_ID_ENUM (see page 316)	
	MODE_JUMPERS_ENUM (see page 316)	
	REG_DIR_ENUM (see page 317)	
	SPI_CSB_ENUM (see page 317)	
	TEST_STATE_ENUM (see page 317)	

Legend

	Enumeration
--	-------------

Structures

	Name	Description
	as_ads1259_registers (see page 297)	This is record as_ads1259_registers.
	bank_info (see page 297)	This is record bank_info.
	bit_string_info (see page 297)	brief
	board_dataset (see page 298)	brief Provides a generic interface to all board datasets. This information is used by several functions in order to determine how to read/write registers, etc.
	board_definition (see page 298)	This is record board_definition.
	command_line (see page 300)	brief Data structure used to decode command line operation.
	command_line_board (see page 300)	This is record command_line_board.
	din_cfg (see page 301)	This is record din_cfg.
	dout_cfg (see page 301)	This is record dout_cfg.
	ec_human_readable (see page 301)	Error code to human readable data structure
	idi_bank_info (see page 302)	list of "bank" names and associated enumerated symbol
	idi_dataset (see page 302)	
	idi_reg_definition (see page 303)	a read only list of register parameters
	ido_bank_info (see page 304)	This is record ido_bank_info.
	ido_dataset (see page 304)	
	ido_reg_definition (see page 305)	
	io_trace (see page 306)	
	io_trace_info (see page 306)	
	IOKERN_TASK_TYPE (see page 307)	
	port_list (see page 309)	
	pt_regs (see page 309)	This is record pt_regs.
	reg_definition (see page 310)	< Generic definition used in IO_Write_U8 (see page 217) and IO_Read_U8 (see page 206)
	spi_cfg (see page 310)	
	spi_dataset (see page 311)	
	spi_status (see page 311)	
	timeval (see page 312)	This is record timeval.

Legend



3.2.2.1 as_ads1259_registers Structure

C++

```
struct as_ads1259_registers {
    uint8_t reg[AS_ADS1259_REGISTER_QTY];
};
```

File

idi.c (see page 373)

Description

This is record as_ads1259_registers.

3.2.2.2 bank_info Structure

C++

```
struct bank_info {
    BANK_ENUM symbol;
    const char * name;
};
```

File

idi.c (see page 373)

Description

This is record bank_info.

3.2.2.3 bit_string_info Structure

C++

```
struct bit_string_info {
    size_t register_offset;
    size_t bit_offset;
    size_t bit_width;
    char * name;
};
```

File

idi.c (see page 373)

Description

brief

3.2.2.4 board_dataset Structure

C++

```
struct board_dataset {
    int board_id;
    BOOL set__suppress_io_activity;
    BOOL quit_application;
    uint16_t base_address;
    int bank_previous;
    BOOL io_simulate;
    BOOL io_report;
    BOOL spi_is_present;
    BOOL fram_cs_default;
    BOOL fram_spi_cs_route_backup;
    BOOL as_cs_default;
    BOOL as_spi_cs_route_backup;
    MODE_JUMPERS_ENUM mode_jumpers;
    const char * svn_revision_string;
    struct spi_dataset spi;
    struct io_trace_info trace;
    struct as_ads1259_registers ads1259;
};
```

File

idi.c (see page 373)

Members

Members	Description
int board_id;	the following must always be at the beginning of this data structure and in the same order
BOOL set__suppress_io_activity;	used by SET BOOL (see page 318) loop_command;
BOOL fram_cs_default;	when fram is used, this will temporarily be used
BOOL as_cs_default;	when analog stick is used, this will temporarily be used

Description

brief Provides a generic interface to all board datasets. This information is used by several functions in order to determine how to read/write registers, etc.

3.2.2.5 board_definition Structure

C++

```
struct board_definition {
    const void * bank_info;
    const void * definition;
```

```
const void * dataset;
const int bank_register_symbol;
const int fpga_data_symbol;
const int fpga_index_symbol;
};
```

File

idi.c (see page 373)

Description

This is record board_definition.

3.2.2.6 buffer_column Enumeration

C++

```
enum buffer_column {
    BUFFER_COLUMN_VALUE = 0,
    BUFFER_COLUMN_TAG = 1
};
```

File

idi.c (see page 373)

Description

brief

3.2.2.7 CLM_BTS Enumeration

C++

```
enum CLM_BTS {
    CLM_BTS_NORMAL = 0,
    CLM_BTS_NOT_REQUIRED = 1,
    CLM_BTS IMPLIED = 2
};
```

File

idi.c (see page 373)

Description

brief

3.2.2.8 command_line Structure

C++

```
struct command_line {
    struct command_line * link;
    int (* cmd_fnc)(int argc, char * argv[]);
    char * name;
    char * help;
};
```

File

idi.c (see page 373)

Members

Members	Description
struct command_line * link;	link to next lower level data structure
int (* cmd_fnc)(int argc, char * argv[]);	function to call to process arguments further
char * name;	name of argument/command word
char * help;	very brief help string associated with command

Description

brief Data structure used to decode command line operation.

3.2.2.9 command_line_board Structure

C++

```
struct command_line_board {
    struct command_line_board * link;
    int (* cmd_fnc)(int board_id, int argc, char * argv[]);
    char * name;
    char * help;
};
```

File

idi.c (see page 373)

Members

Members	Description
struct command_line_board * link;	link to next lower level data structure
int (* cmd_fnc)(int board_id, int argc, char * argv[]);	function to call to process arguments further
char * name;	name of argument/command word
char * help;	very brief help string associated with command

Description

This is record command_line_board.

3.2.2.10 din_cfg Structure

C++

```
struct din_cfg {
    struct {
        BOOL falling_edge;
        BOOL interrupt_enable;
    } chan[IDI_DIN_QTY];
};
```

File

idi.c (see page 373)

Description

This is record din_cfg.

3.2.2.11 dout_cfg Structure

C++

```
struct dout_cfg {
    struct {
        BOOL value;
    } chan[IDO_DO_QTY];
};
```

File

idi.c (see page 373)

Description

This is record dout_cfg.

3.2.2.12 ec_human_readable Structure

C++

```
struct ec_human_readable {
    EC_ENUM error_code;
    const char * message;
};
```

File

idi.c (see page 373)

Description

Error code to human readable data structure

3.2.2.13 idi_bank_info Structure

C++

```
struct idi_bank_info {
    int symbol;
    const char * name;
};
```

File

idi.c (see page 373)

Description

list of "bank" names and associated enumerated symbol

3.2.2.14 idi_dataset Structure

C++

```
struct idi_dataset {
    int board_id;
    BOOL set__suppress_io_activity;
    BOOL quit_application;
    uint16_t base_address;
    IDI_BANK_ENUM bank_previous;
    BOOL io_simulate;
    BOOL io_report;
    BOOL spi_is_present;
    BOOL fram_cs_default;
    BOOL fram_spi_cs_route_backup;
    BOOL as_cs_default;
    BOOL as_spi_cs_route_backup;
    MODE_JUMPERS_ENUM mode_jumpers;
    const char * svn_revision_string;
    struct spi_dataset spi;
    struct io_trace_info trace;
    struct as_ads1259_registers ads1259;
    char message[MESSAGE_SIZE];
    unsigned int irq_number;
    BOOL irq_please_install_handler_request;
    BOOL irq_handler_active;
    size_t irq_quantity;
    size_t volatile irq_count;
    size_t irq_count_previous;
    uint8_t volatile isr_pending_list[IDI_DIN_GROUP_QTY];
```

```

struct din_cfg din_cfg;
struct {
    BOOL used;
    int address;
    IDI_REG_ENUM location;
    uint8_t value;
} reg_init[REGS_INIT_QTY];
};

```

File

idi.c (see page 373)

Members

Members	Description
int board_id;	the following must always be at the beginning of this data structure and in the same order
BOOL set__suppress_io_activity;	used by SET BOOL (see page 318) loop_command;
BOOL io_simulate;	int bank_register_index; /* needs to be initialized at startup!!
BOOL fram_cs_default;	when fram is used, this will temporarily be used
BOOL as_cs_default;	when analog stick is used, this will temporarily be used
char message[MESSAGE_SIZE];	this is a scratch pad area for report generation
unsigned int irq_number;	irq number
BOOL irq.Please_install_handler_request;	MAIN
BOOL irq.handler_active;	MAIN
size_t irq_quantity;	MAIN
size_t volatile irq_count;	ISR
size_t irq_count_previous;	MAIN Isolated digital input specific interrupt parameters <ul style="list-style-type: none"> to be used only by the Interrupt Service Routine (ISR). TODO: modify hardware firmware so that this can be moved to mainline rather than at ISR level.
uint8_t volatile isr_pending_list[IDI_DIN_GROUP_QTY];	ISR
struct din_cfg din_cfg;	dataset portion specific to this particular board

3.2.2.15 idi_reg_definition Structure

C++

```

struct idi_reg_definition {
    IDI_REG_ENUM symbol;
    REG_DIR_ENUM direction;
    IDI_BANK_ENUM bank;
    uint16_t physical_offset;
    char * symbol_name;
    char * acronym;
};

```

File

idi.c (see page 373)

Description

a read only list of register parameters

3.2.2.16 ido_bank_info Structure

C++

```
struct ido_bank_info {
    IDI_BANK_ENUM symbol;
    const char * name;
};
```

File

idi.c (see page 373)

Description

This is record ido_bank_info.

3.2.2.17 ido_dataset Structure

C++

```
struct ido_dataset {
    int board_id;
    BOOL set__suppress_io_activity;
    BOOL quit_application;
    uint16_t base_address;
    IDI_BANK_ENUM bank_previous;
    BOOL io_simulate;
    BOOL io_report;
    BOOL spi_is_present;
    BOOL fram_cs_default;
    BOOL fram_spi_cs_route_backup;
    BOOL as_cs_default;
    BOOL as_spi_cs_route_backup;
    MODE_JUMPERS_ENUM mode_jumpers;
    const char * svn_revision_string;
    struct spi_dataset spi;
    struct io_trace_info trace;
    struct as_ads1259_registers ads1259;
    char message[MESSAGE_SIZE];
    unsigned int irq_number;
    BOOL irq_please_install_handler_request;
    BOOL irq_handler_active;
    size_t irq_quantity;
    size_t volatile irq_count;
    size_t irq_count_previous;
    uint8_t volatile isr_pending_list[IDI_DIN_GROUP_QTY];
```

```

struct dout_cfg dout_cfg;
struct {
    BOOL used;
    int address;
    IDO_REG_ENUM location;
    uint8_t value;
} reg_init[REGS_INIT_QTY];
};

```

File

idi.c (see page 373)

Members

Members	Description
int board_id;	the following must always be at the beginning of this data structure and in the same order
BOOL set__suppress_io_activity;	used by SET BOOL (see page 318) loop_command;
BOOL io_simulate;	int bank_register_index; /* needs to be initialized at startup!!
BOOL fram_cs_default;	when fram is used, this will temporarily be used
BOOL as_cs_default;	when analog stick is used, this will temporarily be used
char message[MESSAGE_SIZE];	this is a scratch pad area for report generation
unsigned int irq_number;	irq number
BOOL irq.Please_install_handler_request;	MAIN
BOOL irq.handler_active;	MAIN
size_t irq_quantity;	MAIN
size_t volatile irq_count;	ISR
size_t irq_count_previous;	MAIN Isolated digital input specific interrupt parameters <ul style="list-style-type: none"> to be used only by the Interrupt Service Routine (ISR). TODO: modify hardware firmware so that this can be moved to mainline rather than at ISR level.
uint8_t volatile isr_pending_list[IDI_DIN_GROUP_QTY];	ISR
struct dout_cfg dout_cfg;	dataset portion specific to this particular board

3.2.2.18 ido_reg_definition Structure

C++

```

struct ido_reg_definition {
    IDO_REG_ENUM symbol;
    REG_DIR_ENUM direction;
    IDO_BANK_ENUM bank;
    uint16_t physical_offset;
    char * symbol_name;
    char * acronym;
};

```

File

idi.c (see page 373)

3.2.2.19 io_trace Structure

C++

```
struct io_trace {
    int error_code;
    enum IO_TRACE_ENUM action;
    int board_id;
    int location;
    size_t line;
    size_t posting;
    uint8_t value;
    const char * function_name;
};
```

File

idi.c (see page 373)

Members

Members	Description
size_t line;	line in source code
size_t posting;	larger numbers => later times

3.2.2.20 IO_TRACE_ENUM Enumeration

C++

```
enum IO_TRACE_ENUM {
};
```

File

idi.c (see page 373)

3.2.2.21 io_trace_info Structure

C++

```
struct io_trace_info {
    size_t begin;
    size_t end;
    size_t index;
    size_t count;
    size_t posting;
    BOOL wrap;
    enum IO_TRACE_START_ENUM start;
```

```
enum IO_TRACE_STOP_ENUM stop;
BOOL active;
char filename[IO_TRACE_FILE_NAME_SIZE];
struct io_trace * buf;
};
```

File

idi.c (see page 373)

Members

Members	Description
size_t posting;	larger number => later times

3.2.2.22 IO_TRACE_START_ENUM Enumeration

C++

```
enum IO_TRACE_START_ENUM {
```

File

idi.c (see page 373)

3.2.2.23 IO_TRACE_STOP_ENUM Enumeration

C++

```
enum IO_TRACE_STOP_ENUM {
```

File

idi.c (see page 373)

3.2.2.24 IOKERN_TASK_TYPE Structure

C++

```
struct IOKERN_TASK_TYPE {
    IOKERN_TASK_FP task_fp;
    IOKERN_HELP_FP help_fp;
    void * dev_id;
    const char * name;
    size_t speed;
    IOKERN_CHAIN_TO_OLD_ENUM chain_to_old;
    unsigned int number;
    size_t sw_int;
    irqreturn_t result;
    unsigned long count;
    IOKERN_ISR_FP old_isr;
```

```
    unsigned char old_state;
};
```

File

idi.c (see page 373)

Members

Members	Description
IOKERN_TASK_FP task_fp;	task to be called for the IRQ
IOKERN_HELP_FP help_fp;	alternate interrupt routing -- timer functions
void * dev_id;	task private data
const char * name;	name of the task
size_t speed;	irq/int fast or slow (slow == push to main (see page 239) loop tasklet)
IOKERN_CHAIN_TO_OLD_ENUM chain_to_old;	See enum above
unsigned int number;	irq/int that this task is mapped too
size_t sw_int;	if software interrupt, then this value is set
irqreturn_t result;	result of the task
unsigned long count;	number of times the IRQ has run
IOKERN_ISR_FP old_isr;	store old ISR here
unsigned char old_state;	PICK MASK copy

3.2.2.25 irqreturn Enumeration

C++

```
enum irqreturn {
    IRQ_NONE,
    IRQ_HANDLED,
    IRQ_WAKE_THREAD
};
```

File

idi.c (see page 373)

Description

The following code is directly from irqreturn.h from the Linux kernel tree. This is being used by the simulator and DOS to emulate Linux driver behavior and specifically resource allocations.

- *
- enum irqreturn
- @IRQ_NONE interrupt was not from this device
- @IRQ_HANDLED interrupt was handled by this device
- @IRQ_WAKE_THREAD handler requests to wake the handler thread

3.2.2.26 port_list Structure

C++

```
struct port_list {
    int address_start;
    int address_end;
    BOOL scan;
    const char * name;
};
```

File

idi.c (see page 373)

3.2.2.27 pt_regs Structure

C++

```
struct pt_regs {
    uint16_t bp;
    union {
        uint32_t edi;
        uint16_t di;
    };
    union {
        uint32_t esi;
        uint16_t si;
    };
    uint16_t ds;
    uint16_t es;
    union {
        uint32_t edx;
        uint16_t dx;
        struct {
            uint8_t l, h;
        } d;
    };
    union {
        uint32_t ecx;
        uint16_t cx;
        struct {
            uint8_t l, h;
        } c;
    };
    union {
        uint32_t ebx;
        uint16_t bx;
        struct {
            uint8_t l, h;
        } b;
    };
    union {
        uint32_t eax;
        uint16_t ax;
        struct {
            uint16_t l, h;
        } aw;
    };
};
```

```

struct {
    uint8_t l, h;
} a;
};
uint16_t ip;
uint16_t cs;
uint16_t flags;
};

```

File

idi.c ([see page 373](#))

Description

This is record pt_regs.

3.2.2.28 reg_definition Structure

C++

```

struct reg_definition {
    int symbol;
    REG_DIR_ENUM direction;
    int bank;
    uint16_t physical_offset;
    char * symbol_name;
    char * acronym;
};

```

File

idi.c ([see page 373](#))

Description

< Generic definition used in IO_Write_U8 ([see page 217](#)) and IO_Read_U8 ([see page 206](#))

3.2.2.29 spi_cfg Structure

C++

```

struct spi_cfg {
    BOOL sdio_wrap;
    BOOL sdo_polarity;
    BOOL mdi_polarity;
    BOOL sclk_phase;
    BOOL sclk_polarity;
    BOOL chip_select_route_ch1;
    SPI_CSB_ENUM chip_select_behavior;
    uint8_t end_cycle_delay;
    uint16_t half_clock_interval;
    double clock_hz;
    double end_delay_ns;
};

```

File

idi.c (see page 373)

Members

Members	Description
BOOL sdio_wrap;	< SDIO_WRAP
BOOL sdo_polarity;	< SDO_POL
BOOL sdi_polarity;	< SDI_POL
BOOL sclk_phase;	< SCLK_PHA
BOOL sclk_polarity;	< SCLK_POL
BOOL chip_select_route_ch1;	< CS_ROUTE
SPI_CS_B_ENUM chip_select_behavior;	< CSB[2:0]
uint8_t end_cycle_delay;	< ECD[7:0]
uint16_t half_clock_interval;	< HCI[11:0]
double clock_hz;	< if nonzero, code will compute half_clock_interval
double end_delay_ns;	< if nonzero, code will compute end_cycle_dealy

3.2.2.30 spi_dataset Structure

C++

```
struct spi_dataset {
    uint16_t id;
    struct spi_cfg cfg;
    uint8_t fram_block[FRAM_BLOCK_SIZE];
    uint8_t tx_buffer[SPI_BLOCK_SIZE];
    uint8_t rx_buffer[SPI_BLOCK_SIZE];
};
```

File

idi.c (see page 373)

Members

Members	Description
uint8_t fram_block[FRAM_BLOCK_SIZE];	this is a scratch pad for FRAM transactions
uint8_t tx_buffer[SPI_BLOCK_SIZE];	used by: IDI_CMD_SPI_Data()

3.2.2.31 spi_status Structure

C++

```
struct spi_status {
    BOOL tx_status;
    BOOL full;
    BOOL empty;
    int fifo_size;
```

```
    int fifo_count;
};
```

File

idi.c (see page 373)

Members

Members	Description
BOOL tx_status;	meaning this is specific to TX status, not RX status

3.2.2.32 timeval Structure

C++

```
struct timeval {
    long tv_sec;
    long tv_usec;
};
```

File

stopwatch.h (see page 580)

Members

Members	Description
long tv_sec;	seconds
long tv_usec;	microseconds

Description

This is record timeval.

3.2.2.33 BANK_ENUM Enumeration

C++

```
typedef enum {
} BANK_ENUM;
```

File

idi.c (see page 373)

Description

This is type BANK_ENUM.

3.2.2.34 EC_ENUM Enumeration

C++

```
typedef enum {  
} EC_ENUM;
```

File

idi.c (see page 373)

Description

EC = Error Code

3.2.2.35 IDI_BANK_ENUM Enumeration

C++

```
typedef enum {  
} IDI_BANK_ENUM;
```

File

idi.c (see page 373)

Description

This is type IDI_BANK_ENUM.

3.2.2.36 IDI_REG_ENUM Enumeration

C++

```
typedef enum {  
} IDI_REG_ENUM;
```

File

idi.c (see page 373)

Description

brief Once the register location (i.e. logical_address or row) is known, then all other parameters associated with that register can be looked up including physical_offset, bank and legal read/write access.

3.2.2.37 IDO_BANK_ENUM Enumeration

C++

```
typedef enum {
} IDO_BANK_ENUM;
```

File

idi.c (see page 373)

Description

This is type IDO_BANK_ENUM.

3.2.2.38 IDO_REG_ENUM Enumeration

C++

```
typedef enum {
} IDO_REG_ENUM;
```

File

idi.c (see page 373)

3.2.2.39 IOKERN_CHAIN_TO_OLD_ENUM Enumeration

C++

```
typedef enum {
    IOKERN_CHAIN_TO_OLD_OFF = 0,
    IOKERN_CHAIN_TO_OLD_TIMER = 1,
    IOKERN_CHAIN_TO_OLD_NORMAL = 2
} IOKERN_CHAIN_TO_OLD_ENUM;
```

File

idi.c (see page 373)

Members

Members	Description
IOKERN_CHAIN_TO_OLD_OFF = 0	if help_fp==NULL then this is always the case
IOKERN_CHAIN_TO_OLD_TIMER = 1	chain to the old timer, but only when iokern_timer_periodic_chain_count reaches zero.
IOKERN_CHAIN_TO_OLD_NORMAL = 2	chain to old interrupt

3.2.2.40 IOKERN_IRQ_ENUM Enumeration

C++

```
typedef enum {
    IOKERN_IRQ_NONE = UINT_MAX,
    IOKERN_IRQ_0 = 0,
    IOKERN_IRQ_1 = 1,
    IOKERN_IRQ_2 = 2,
    IOKERN_IRQ_3 = 3,
    IOKERN_IRQ_4 = 4,
    IOKERN_IRQ_5 = 5,
    IOKERN_IRQ_6 = 6,
    IOKERN_IRQ_7 = 7,
    IOKERN_IRQ_8 = 8,
    IOKERN_IRQ_9 = 9,
    IOKERN_IRQ_10 = 10,
    IOKERN_IRQ_11 = 11,
    IOKERN_IRQ_12 = 12,
    IOKERN_IRQ_13 = 13,
    IOKERN_IRQ_14 = 14,
    IOKERN_IRQ_15 = 15,
    IOKERN_INT_33 = 0x33
} IOKERN_IRQ_ENUM;
```

File

idi.c (see page 373)

Members

Members	Description
IOKERN_IRQ_0 = 0	timer
IOKERN_IRQ_1 = 1	keyboard or PS/2
IOKERN_IRQ_2 = 2	reserved for 2nd 8259 (IRQ8-15) or IRQ2 for XT
IOKERN_IRQ_3 = 3	pc104 - typically COM2,4
IOKERN_IRQ_4 = 4	pc104 - typically COM1,3
IOKERN_IRQ_5 = 5	pc104 - typically LPT2 or sound
IOKERN_IRQ_6 = 6	pc104 - typically floppy
IOKERN_IRQ_7 = 7	pc104 - typically LPT1
IOKERN_IRQ_8 = 8	real-time clock
IOKERN_IRQ_10 = 10	pc104
IOKERN_IRQ_11 = 11	pc104
IOKERN_IRQ_12 = 12	pc104 - typically, PS2 mouse
IOKERN_IRQ_13 = 13	Numerical processor unit
IOKERN_IRQ_14 = 14	pc104 - primary IDE
IOKERN_IRQ_15 = 15	pc104 - secondary IDE software interrupts

3.2.2.41 IOKERN_TASK_ID_ENUM Enumeration

C++

```
typedef enum {
    IOKERN_TASK_ID_NONE = -1,
    IOKERN_TASK_ID_IRQ0 = 0,
    IOKERN_TASK_ID_IRQ1 = 1,
    IOKERN_TASK_ID_IRQ2 = 2,
    IOKERN_TASK_ID_IRQ3 = 3,
    IOKERN_TASK_ID_IRQ4 = 4,
    IOKERN_TASK_ID_IRQ5 = 5,
    IOKERN_TASK_ID_IRQ6 = 6,
    IOKERN_TASK_ID_IRQ7 = 7,
    IOKERN_TASK_ID_IRQ8 = 8,
    IOKERN_TASK_ID_IRQ9 = 9,
    IOKERN_TASK_ID_IRQ10 = 10,
    IOKERN_TASK_ID_IRQ11 = 11,
    IOKERN_TASK_ID_IRQ12 = 12,
    IOKERN_TASK_ID_IRQ13 = 13,
    IOKERN_TASK_ID_IRQ14 = 14,
    IOKERN_TASK_ID_IRQ15 = 15,
    IOKERN_TASK_ID_INT33 = 16
} IOKERN_TASK_ID_ENUM;
```

File

idi.c (see page 373)

Members

Members	Description
IOKERN_TASK_ID_INT33 = 16	mouse function calls

3.2.2.42 MODE_JUMPERS_ENUM Enumeration

C++

```
typedef enum {
    MODE_JUMPERS_0 = 0,
    MODE_JUMPERS_1 = 1,
    MODE_JUMPERS_2 = 2,
    MODE_JUMPERS_3 = 3
} MODE_JUMPERS_ENUM;
```

File

idi.c (see page 373)

Members

Members	Description
MODE_JUMPERS_0 = 0	legacy

3.2.2.43 REG_DIR_ENUM Enumeration

C++

```
typedef enum {
    REG_DIR_NONE = 0x00,
    REG_DIR_READ = 0x01,
    REG_DIR_WRITE = 0x02,
    REG_DIR_READ_WRITE = 0x03
} REG_DIR_ENUM;
```

File

idi.c (see page 373)

3.2.2.44 SPI_CSB_ENUM Enumeration

C++

```
typedef enum {
    CSB_SOFTWARE = 0,
    CSB_BUFFER = 1,
    CSB_uint8_t = 2,
    CSB_uint16_t = 3
} SPI_CSB_ENUM;
```

File

idi.c (see page 373)

3.2.2.45 TEST_STATE_ENUM Enumeration

C++

```
typedef enum {
    TEST_STATE_INIT = 0x000,
    TEST_STATE_READY = 0x100,
    TEST_STATE_ACTIVE = 0x200,
    TEST_STATE_COMPLETE = 0xF00,
    TEST_STATE_QUIT_ERROR = 0xF10,
    TEST_STATE_QUIT_USER = 0xF20,
    TEST_STATE_QUIT = 0xFFFF
} TEST_STATE_ENUM;
```

File

idi.c (see page 373)

3.2.3 Types

The following table lists types in this documentation.

Types

Name	Description
BOOL (see page 318)	brief Boolean logic definitions
int16_t (see page 318)	DOS only
int32_t (see page 319)	DOS only
int8_t (see page 319)	This is type int8_t.
IOKERN_DOS_VECTOR_TYPE (see page 319)	
IOKERN_HELP_FP (see page 319)	This is type IOKERN_HELP_FP.
IOKERN_ISR_FP (see page 320)	This is type IOKERN_ISR_FP.
IOKERN_TASK_FP (see page 320)	
irqreturn_t (see page 320)	irqreturn_t is directly from Linux kernel
STOPWATCH_TIMEVAL_TYPE (see page 321)	This is type STOPWATCH_TIMEVAL_TYPE.
uint16_t (see page 321)	DOS only
uint32_t (see page 321)	DOS only
uint8_t (see page 321)	brief The C89 compiler is typically void of these definitions, so we include them here. Defines specific data width information. The idea is to make this target independent.

3.2.3.1 BOOL Type

C++

```
typedef int BOOL;
```

File

idi.c (see page 373)

Description

brief Boolean logic definitions

3.2.3.2 int16_t Type

C++

```
typedef int int16_t;
```

File

idi.c (see page 373)

Description

DOS only

3.2.3.3 int32_t Type

C++

```
typedef long int32_t;
```

File

idi.c (see page 373)

Description

DOS only

3.2.3.4 int8_t Type

C++

```
typedef char int8_t;
```

File

idi.c (see page 373)

Description

This is type int8_t.

3.2.3.5 IOKERN_DOS_VECTOR_TYPE Type

C++

```
typedef void interrupt (* IOKERN_DOS_VECTOR_TYPE)();
```

File

idi.c (see page 373)

3.2.3.6 IOKERN_HELP_FP Type

C++

```
typedef void (* IOKERN_HELP_FP)(IOKERN_TASK_ID_ENUM id, IOKERN_IRQ_ENUM irq, struct pt_regs * regs);
```

File

idi.c (see page 373)

Description

This is type IOKERN_HELP_FP.

3.2.3.7 IOKERN_ISR_FP Type

C++

```
typedef void INTERRUPT (* IOKERN_ISR_FP)(struct pt_regs r);
```

File

idi.c (see page 373)

Description

This is type IOKERN_ISR_FP.

3.2.3.8 IOKERN_TASK_FP Type

C++

```
typedef irqreturn_t (* IOKERN_TASK_FP)(int irq, void * dev_id, struct pt_regs * regs);
```

File

idi.c (see page 373)

3.2.3.9 irqreturn_t Type

C++

```
typedef enum irqreturn irqreturn_t;
```

File

idi.c (see page 373)

Description

irqreturn_t is directly from Linux kernel

3.2.3.10 STOPWATCH_TIMEVAL_TYPE Type

C++

```
typedef struct timeval STOPWATCH_TIMEVAL_TYPE;
```

File

stopwatch.h (see page 580)

Description

This is type STOPWATCH_TIMEVAL_TYPE.

3.2.3.11 uint16_t Type

C++

```
typedef unsigned int uint16_t;
```

File

idi.c (see page 373)

Description

DOS only

3.2.3.12 uint32_t Type

C++

```
typedef unsigned long uint32_t;
```

File

idi.c (see page 373)

Description

DOS only

3.2.3.13 uint8_t Type

C++

```
typedef unsigned char uint8_t;
```

File

idi.c ([see page 373](#))

Description

brief The C89 compiler is typically void of these definitions, so we include them here. Defines specific data width information. The idea is to make this target independent.

3.2.4 Variables

The following table lists variables in this documentation.

Variables

Name	Description
ADS1259_CLOCK_MHZ (see page 323)	static BOOL (see page 318) as_checksum = false;
ADS1259_CLOCK_TOLERANCE (see page 323)	2
ADS1259_FSC_MAX (see page 324)	0x00800000 => x2
ADS1259_FSC_MIN (see page 324)	0x00000000 => x0
ADS1259_OFC_MAX (see page 324)	0x007FFFFF
ADS1259_OFC_MIN (see page 324)	0xFF800001
as_bit_string (see page 325)	brief
as_register_name (see page 325)	brief
buffer (see page 325)	This is variable buffer.
buffer_index (see page 326)	This is variable buffer_index.
cmd_as (see page 326)	brief
cmd_fram (see page 327)	brief
cmd_iai16 (see page 327)	brief
cmd_idi48 (see page 327)	brief
cmd_idi48_test (see page 328)	brief
cmd_ido48 (see page 328)	brief
cmd_ido48_test (see page 328)	brief
cmd_loop (see page 329)	brief
cmd_prefix (see page 329)	brief
cmd_set (see page 329)	brief
cmd_spi (see page 330)	brief
cmd_top (see page 330)	brief
cmd_top_board_type_required (see page 331)	This is variable cmd_top_board_type_required.
cmd_trace (see page 331)	brief
ec_unknown (see page 331)	brief Translates an error code into a human readable message. Excerpt from AES advanced Linux library/driver. COPYRIGHT NOTICE Copyright (c) 2012 by Apex Embedded Systems. param[in] error_code The error code to be translated into a human readable message

global_board_id (see page 332)	This is variable global_board_id.
global_io_trace_buf (see page 332)	
global_io_trace_enable (see page 332)	This is variable global_io_trace_enable.
global_io_trace_name (see page 332)	This is variable global_io_trace_name.
global_io_trace_start_name (see page 333)	This is variable global_io_trace_start_name.
global_io_trace_stop_name (see page 333)	This is variable global_io_trace_stop_name.
global_irq.Please_install_handler_request (see page 333)	This is variable global_irq.Please_install_handler_request.
global_loop_command (see page 334)	This is variable global_loop_command.
global_loop_count (see page 334)	This is variable global_loop_count.
global_loop_count_counter (see page 334)	This is variable global_loop_count_counter.
global_loop_delay_ms (see page 334)	This is variable global_loop_delay_ms.
global_loop_space (see page 335)	This is variable global_loop_space.
idi_ds (see page 335)	Global data structure which is restored during initialization and saved during application termination.
ido_ds (see page 335)	
iokern_isr_table (see page 336)	
iokern_task (see page 336)	This is variable iokern_task.
lpm_lx800_port_list (see page 336)	This is variable lpm_lx800_port_list.
svn_revision_string (see page 337)	Software revision as determined by subversion commits

3.2.4.1 ADS1259_CLOCK_MHZ Variable

C++

```
const double ADS1259_CLOCK_MHZ = 7.3728e6;
```

File

idi.c (see page 373)

Description

static BOOL (see page 318) as_checksum = false;

3.2.4.2 ADS1259_CLOCK_TOLERANCE Variable

C++

```
const double ADS1259_CLOCK_TOLERANCE = 0.02;
```

File

idi.c (see page 373)

Description

2

3.2.4.3 ADS1259_FSC_MAX Variable

C++

```
const int32_t ADS1259_FSC_MAX = 8388608L;
```

File

idi.c (see page 373)

Description

0x00800000 => x2

3.2.4.4 ADS1259_FSC_MIN Variable

C++

```
const int32_t ADS1259_FSC_MIN = 0L;
```

File

idi.c (see page 373)

Description

0x00000000 => x0

3.2.4.5 ADS1259_OFC_MAX Variable

C++

```
const int32_t ADS1259_OFC_MAX = 8388607L;
```

File

idi.c (see page 373)

Description

0x007FFFFF

3.2.4.6 ADS1259_OFC_MIN Variable

C++

```
const int32_t ADS1259_OFC_MIN = -8388607L;
```

File

idi.c (see page 373)

Description

0xFF800001

3.2.4.7 as_bit_string Variable

C++

```
struct bit_string_info as_bit_string[] = { { AS_AMS1259_CONFIG0, 0, 1, "spi" }, { AS_AMS1259_CONFIG0, 2, 1, "rbias" }, { AS_AMS1259_CONFIG0, 4, 2, "id" }, { AS_AMS1259_CONFIG1, 0, 3, "delay" }, { AS_AMS1259_CONFIG1, 3, 1, "extref" }, { AS_AMS1259_CONFIG1, 4, 1, "sinc2" }, { AS_AMS1259_CONFIG1, 6, 1, "chksum" }, { AS_AMS1259_CONFIG1, 7, 1, "flag" }, { AS_AMS1259_CONFIG2, 0, 3, "dr" }, { AS_AMS1259_CONFIG2, 4, 1, "pulse" }, { AS_AMS1259_CONFIG2, 5, 1, "syncout" }, { AS_AMS1259_CONFIG2, 6, 1, "extclk" }, { AS_AMS1259_CONFIG2, 7, 1, "drdy_b" }, { AS_AMS1259_REG_QTY, 0, 0, NULL } };
```

File

idi.c (see page 373)

Description

brief

3.2.4.8 as_register_name Variable

C++

```
char * as_register_name[] = { "config0", "config1", "config2", "ofc0", "ofc1", "ofc2", "fsc0", "fsc1", "fsc2", NULL };
```

File

idi.c (see page 373)

Description

brief

3.2.4.9 buffer Variable

C++

```
int32_t buffer[BUFFER_COLUMN][BUFFER_LENGTH];
```

File

idi.c (see page 373)

Description

This is variable buffer.

3.2.4.10 buffer_index Variable

C++

```
size_t buffer_index;
```

File

idi.c (see page 373)

Description

This is variable buffer_index.

3.2.4.11 cmd_as Variable

C++

```
struct command_line_board cmd_as[] = { { NULL, CMD_AS_Wakeup, "wakeup", "ADS1259 exit sleep mode" }, { NULL, CMD_AS_Sleep, "sleep", "ADS1259 enter sleep mode" }, { NULL, CMD_AS_Reset, "reset", "ADS1259 register and filter reset" }, { NULL, CMD_AS_Start, "start", "ADS1259 start conversion(s)" }, { NULL, CMD_AS_Stop, "stop", "ADS1259 stop conversions" }, { NULL, CMD_AS_Rdatac, "rdatac", "ADS1259 set read data continuous mode" }, { NULL, CMD_AS_Sdatac, "sdatac", "ADS1259 stop read data continuous mode" }, { NULL, CMD_AS_Rdata, "rdata", "ADS1259 read data in stop continuous mode" }, { NULL, CMD_AS_RReg, "rreg", "ADS1259 read N registers, rreg <address> <count>" }, { NULL, CMD_AS_WReg, "wreg", "ADS1259 write N registers, wreg <address> <value> [<value>]" }, { NULL, CMD_AS_Ofscal, "ofscal", "ADS1259 perform offset calibration, apply zero input" }, { NULL, CMD_AS_Gancal, "gancal", "ADS1259 perform full-scale calibration, apply full-scale" }, { NULL, CMD_AS_Reg_Write_By_Name, "wrn", "Write one or more registers by name, wrn <name> <value>" }, { NULL, CMD_AS_Calibration_Gain, "fsc", "read/write fullscale calibration" }, { NULL, CMD_AS_Calibration_Offset, "ofc", "read/write offset calibration" }, { NULL, CMD_AS_Register_Save, "rsave", "save all registers to a file" }, { NULL, CMD_AS_Register_Load, "rload", "restore all registers from file" }, { NULL, CMD_AS_Parameter, "param", "read/write parameter" }, { NULL, NULL, NULL, NULL } };
```

File

idi.c (see page 373)

Description

brief

3.2.4.12 cmd_fram Variable

C++

```
struct command_line_board cmd_fram[] = { { NULL, CMD_FRAM_Dump, "dump", "params: <address> <length>" }, { NULL, CMD_FRAM_Save, "save", "params: <address> <length> <binary destination file>" }, { NULL, CMD_FRAM_Load, "load", "params: <address> <binary source file name>" }, { NULL, CMD_FRAM_Write, "write", "params: <address> <data list: bytes/characters/strings>" }, { NULL, CMD_FRAM_Init, "init", "params: [byte/character] [byte/character] ..." }, { NULL, CMD_FRAM_WREN, "wren", "WRite Enable Latch Set" }, { NULL, CMD_FRAM_WRDI, "wrdi", "WRite Disable" }, { NULL, CMD_FRAM_RDSR, "rdsr", "ReaD Status Register" }, { NULL, CMD_FRAM_WRSR, "wrsr", "WRite Status Register. Params: <status>" }, { NULL, CMD_FRAM_RDID, "rdid", "ReaD ID Register" }, { NULL, NULL, NULL, NULL } };
```

File

idi.c (see page 373)

Description

brief

3.2.4.13 cmd_iai16 Variable

C++

```
struct command_line_board cmd_iai16[] = { { NULL, CMD_IAI16_AI_ID, "id", "params: none. Reports the AI board/component ID." }, { NULL, CMD_IAI16_AI_Channel, "chan", "params: <channel>" }, { NULL, CMD_IAI16_AI_All, "all", "gets all analog inputs" }, { cmd_ido48_test, CMD_IAI16_Test, "test", "semi-auto test functions" }, { NULL, NULL, NULL, NULL } };
```

File

idi.c (see page 373)

Description

brief

3.2.4.14 cmd_idi48 Variable

C++

```
struct command_line_board cmd_idi48[] = { { NULL, CMD_IDI48_DIN_ID, "id", "params: none. Reports the DIN board/component ID." }, { NULL, CMD_IDI48_DIN_Channel, "chan", "params: <channel>" }, { NULL, CMD_IDI48_DIN_Group, "group", "params: [<group_channel | all>]" }, { NULL, CMD_IDI48_DIN_All, "all", "reports all digital inputs in binary and hex" }, { cmd_idi48_test, CMD_IDI48_DIN_Test, "test", "semi-auto test functions" }, { NULL, NULL, NULL, NULL } };
```

File

idi.c (see page 373)

Description

brief

3.2.4.15 cmd_idi48_test Variable

C++

```
struct command_line_board cmd_idi48_test[] = { { NULL, CMD_IDI48_DIN_Test_Value, "value",  
"loop, until all have toggled" }, { NULL, CMD_IDI48_DIN_Test_RE, "re", "loop, rising-edge  
detect (led perspective)" }, { NULL, CMD_IDI48_DIN_Test_FE, "fe", "loop, falling-edge detect  
(led perspective)" }, { NULL, CMD_IDI48_DIN_Test_Interrupt, "interrupt", "loop, test interrupt  
on all inputs" }, { NULL, NULL, NULL, NULL }, };
```

File

idi.c (see page 373)

Description

brief

3.2.4.16 cmd_ido48 Variable

C++

```
struct command_line_board cmd_ido48[] = { { NULL, CMD_IDO48_DO_ID, "id", "params: none.  
Reports the DOUT board/component ID." }, { NULL, CMD_IDO48_DO_Channel, "chan", "params:  
<channel> <value>" }, { NULL, CMD_IDO48_DO_Group, "group", "params: [<group_channel | all>]"  
, { NULL, CMD_IDO48_DO_All, "all", "gets all digital outputs in binary and hex" }, {  
cmd_ido48_test, CMD_IDO48_Test, "test", "semi-auto test functions" }, { NULL, NULL, NULL,  
NULL } };
```

File

idi.c (see page 373)

Description

brief

3.2.4.17 cmd_ido48_test Variable

C++

```
struct command_line_board cmd_ido48_test[] = { { NULL, CMD_IDO48_DOUT_Test_One_Hot, "onehot",  
"toggling one at a time forever" }, { NULL, CMD_IDO48_DOUT_Test_Alternate, "alternate",  
"alternate pattern at all ports forever" }, { NULL, CMD_IDO48_IDI48_Loopback, "loopback",  
"IDI48 & IDO48 Loopback test" }, { NULL, NULL, NULL, NULL }, };
```

File

idi.c (see page 373)

Description

brief

3.2.4.18 cmd_loop Variable

C++

```
struct command_line cmd_loop[] = { { NULL, Command_Line_Loop_Count, "count", "loop for N counts" }, { NULL, Command_Line_Loop_Delay, "delay", "loop with delay <milliseconds>" }, { NULL, Command_Line_Loop_Space, "space", "loop each time space bar pressed" }, { NULL, NULL, NULL, NULL } };
```

File

idi.c (see page 373)

Description

brief

3.2.4.19 cmd_prefix Variable

C++

```
struct command_line cmd_prefix[] = { { NULL, Command_Line_Prefix_Irq, "irq", "allows commands to handle interrupts" }, { cmd_loop, Command_Line_Prefix_Loop, "loop", "any command loops until key pressed" }, { NULL, Command_Line_Prefix_Trace, "trace", "enable trace functionality" }, { NULL, NULL, NULL, NULL } };
```

File

idi.c (see page 373)

Description

brief

3.2.4.20 cmd_set Variable

C++

```
struct command_line_board cmd_set[] = { { NULL, CMD_Main_Base, "base", "params: [<address>]" }, { NULL, CMD_Main_IO_Behavior, "io", "params: [<simulate>/<report>]" }, { NULL, CMD_Main_Irq_Number, "irq", "params: [<irq_number 0 to 15>]" }, { NULL, CMD_Main_I_Count, "iqty", "params: [<number>]. Number of interrupts." }, { NULL, CMD_Main_FRAM_CS, "framcs", "params: [<0/1>]. FRAM SPI chip select channel." }, { NULL, CMD_Main_AnalogStick_CS, "aics", "params: [<0/1>]. Analog stick CS pin." } };
```

```
"params: [<0/1>]. Analog Stick SPI chip select channel." }, { NULL, CMD__AS_Parameter, "as",
"Analog Stick register or parameter read/write" }, { cmd_trace, CMD__Main_Trace, "trace",
"params: [<start/stop/file>]." }, { NULL, CMD__Main_Mode_Jumpers, "mode", "params: [<0/1/2/3>].
Mode jumper settings" }, { NULL, NULL, NULL, NULL } , };
```

File

idi.c (see page 373)

Description

brief

3.2.4.21 cmd_spi Variable

C++

```
struct command_line_board cmd_spi[] = { { NULL, CMD__SPI_ID, "id", "wishbone id: params: none"
}, { NULL, CMD__SPI_Config_Get, "cfg", "config dump: params: none" }, { NULL,
CMD__SPI_Config_Clock_Hz, "clk", "clk: params: [<clock freq in hertz>]" }, { NULL,
CMD__SPI_Config_End_Cycle_Delay_Sec, "ecd", "end delay: params: [<time in seconds>]" }, {
NULL, CMD__SPI_Config_Mode, "mode", "mode: params: [<0/1/2/3>]" }, { NULL,
CMD__SPI_Config_SD1_Polarity, "sdi", "sdi pol: params: [<true/1/false/0>]" }, { NULL,
CMD__SPI_Config_SDO_Polarity, "sdo", "sdo pol: params: [<true/1/false/0>]" }, { NULL,
CMD__SPI_Config_SDIO_Wrap, "wrap", "sdo-->sdi: params: [<true/1/false/0>]" }, { NULL,
CMD__SPI_Config_Chip_Select_Behavior, "csb", "chip select behavior: params:
[0/1/2/3/software/buffer/uint8_t/uint16_t]" }, { NULL, CMD__SPI_Config_Chip_Select_Route,
"csr", "chip select routing: params: [<true/1/false/0>]" }, { NULL, CMD__SPI_Status, "status",
"status of both TX and RX buffers" }, { NULL, CMD__SPI_Data, "data", "read/write: params:
<data list: bytes/characters/strings>" }, { NULL, CMD__SPI_FIFO, "fifo", "fifo r/w: params:
<data list: bytes/characters/strings>" }, { NULL, CMD__SPI_Commit, "commit", "causes spi
transactions to start" }, { NULL, NULL, NULL, NULL } };
```

File

idi.c (see page 373)

Description

brief

3.2.4.22 cmd_top Variable

C++

```
struct command_line_board cmd_top[] = { CMD_MAIN_LIST( CMD_MAIN_EXTRACT_COMMANDS ) };
```

File

idi.c (see page 373)

Description

brief

3.2.4.23 cmd_top_board_type_required Variable

C++

```
BOOL cmd_top_board_type_required[] = { CMD_MAIN_LIST( CMD_MAIN_EXTRACT_BOARD_TYPE_REQUIRED ) };
```

File

idi.c (see page 373)

Description

This is variable cmd_top_board_type_required.

3.2.4.24 cmd_trace Variable

C++

```
struct command_line_board cmd_trace[] = { { NULL, CMD_Trace_Start, "start", "begin trace on: < " IO_TRACE_START_DEFINITION( IO_TRACE_START_STOP_EXTRACT_NAME_HELP_LIST ) ">" }, { NULL, CMD_Trace_Stop, "stop", "stop trace on: < " IO_TRACE_STOP_DEFINITION( IO_TRACE_START_STOP_EXTRACT_NAME_HELP_LIST ) ">" }, { NULL, CMD_Trace_File, "file", "file name up to 32-characters" }, { NULL, NULL, NULL, NULL } };
```

File

idi.c (see page 373)

Description

brief

3.2.4.25 ec_unknown Variable

C++

```
const char ec_unknown[] = "unknown error code";
```

File

idi.c (see page 373)

Returns

a human readable string representing a very brief description of the error code.

Description

brief Translates an error code into a human readable message.

Excerpt from AES advanced Linux library/driver. COPYRIGHT NOTICE Copyright (c) 2012 by Apex Embedded Systems.

param[in] error_code The error code to be translated into a human readable message

3.2.4.26 global_board_id Variable

C++

```
int global_board_id = ID_IDI48;
```

File

idi.c (see page 373)

Description

This is variable global_board_id.

3.2.4.27 global_io_trace_buf Variable

C++

```
struct io_trace global_io_trace_buf[IO_TRACE_SIZE];
```

File

idi.c (see page 373)

3.2.4.28 global_io_trace_enable Variable

C++

```
BOOL global_io_trace_enable;
```

File

idi.c (see page 373)

Description

This is variable global_io_trace_enable.

3.2.4.29 global_io_trace_name Variable

C++

```
const char * global_io_trace_name[] = { IO_TRACE_DEFINITION( IO_TRACE_EXTRACT_NAME ) NULL };
```

File

idi.c (see page 373)

Description

This is variable global_io_trace_name.

3.2.4.30 global_io_trace_start_name Variable

C++

```
const char * global_io_trace_start_name[] = { IO_TRACE_START_DEFINITION( IO_TRACE_START_STOP_EXTRACT_NAME ) NULL };
```

File

idi.c (see page 373)

Description

This is variable global_io_trace_start_name.

3.2.4.31 global_io_trace_stop_name Variable

C++

```
const char * global_io_trace_stop_name[] = { IO_TRACE_STOP_DEFINITION( IO_TRACE_START_STOP_EXTRACT_NAME ) NULL };
```

File

idi.c (see page 373)

Description

This is variable global_io_trace_stop_name.

3.2.4.32 global_irq_please_install_handler_request Variable

C++

```
BOOL global_irq_please_install_handler_request = false;
```

File

idi.c (see page 373)

Description

This is variable global_irq_please_install_handler_request.

3.2.4.33 global_loop_command Variable

C++

```
BOOL global_loop_command = false;
```

File

idi.c (see page 373)

Description

This is variable global_loop_command.

3.2.4.34 global_loop_count Variable

C++

```
int global_loop_count = 0;
```

File

idi.c (see page 373)

Description

This is variable global_loop_count.

3.2.4.35 global_loop_count_counter Variable

C++

```
int global_loop_count_counter = 0;
```

File

idi.c (see page 373)

Description

This is variable global_loop_count_counter.

3.2.4.36 global_loop_delay_ms Variable

C++

```
int global_loop_delay_ms = 0;
```

File

idi.c (see page 373)

Description

This is variable global_loop_delay_ms.

3.2.4.37 global_loop_space Variable

C++

```
BOOL global_loop_space = false;
```

File

idi.c (see page 373)

Description

This is variable global_loop_space.

3.2.4.38 idi_ds Variable

C++

```
struct idi_dataset idi_ds;
```

File

idi.c (see page 373)

Description

Global data structure which is restored during initialization and saved during application termination.

3.2.4.39 ido_ds Variable

C++

```
struct ido_dataset ido_ds;
```

File

idi.c (see page 373)

3.2.4.40 iokern_isr_table Variable

C++

```
IOKERN_ISR_FP iokern_isr_table[IOKERN_TASK_QTY] = { IOKern_ISR0, IOKern_ISR1, IOKern_ISR2,
IOKern_ISR3, IOKern_ISR4, IOKern_ISR5, IOKern_ISR6, IOKern_ISR7, IOKern_ISR8, IOKern_ISR9,
IOKern_ISR10, IOKern_ISR11, IOKern_ISR12, IOKern_ISR13, IOKern_ISR14, IOKern_ISR15, NULL };
```

File

idi.c (see page 373)

3.2.4.41 iokern_task Variable

C++

```
IOKERN_TASK_TYPE iokern_task[IOKERN_TASK_QTY];
```

File

idi.c (see page 373)

Description

This is variable iokern_task.

3.2.4.42 lpm_lx800_port_list Variable

C++

```
const struct port_list lpm_lx800_port_list[] = { { 0x000, 0x00F, 0, "8237DMA Controller #1" },
{ 0x010, 0x01F, 1, "Free" }, { 0x020, 0x021, 0, "8259 PIC #1" }, { 0x022, 0x03F, 1, "Free" },
{ 0x040, 0x043, 0, "8254 PIT" }, { 0x044, 0x04D, 1, "Free" }, { 0x04E, 0x04F, 0, "Reserved for
on, Oxbard configuration" }, { 0x050, 0x05F, 1, "Free" }, { 0x060, 0x06F, 0, "8042 Keyboard
/ Mouse Controller" }, { 0x070, 0x07F, 0, "CMOS RAM, Clock / Calendar" }, { 0x080, 0x09F, 0,
"DMA Page Registers" }, { 0x0A0, 0x0BF, 0, "8259 PIC #2" }, { 0x0C0, 0x0DF, 0, "8237 DMA
Controller #2" }, { 0x0E0, 0x0EF, 1, "Free" }, { 0x0F0, 0x0F1, 0, "Math Co, 0xprocessor
Control" }, { 0x0F2, 0x0F7, 1, "Free" }, { 0x0F8, 0x0FF, 0, "Math Co, 0xprocessor" },
{ 0x100, 0x102, 0, "Video Controllers" }, { 0x103, 0x11F, 1, "Free" }, { 0x120, 0x12F, 0,
"Digital I/O" }, { 0x130, 0x14F, 1, "Free" }, { 0x150, 0x150, 0, "Reserved for on, Oxbard
configuration" }, { 0x151, 0x1CF, 1, "Free" }, { 0x1D0, 0x1DF, 0, "Legacy Watchdog (1D0,
0xEnabled; 1D8 , 0x Pet)" }, { 0x1E8, 0x1EB, 0, "Reserved for on, Oxbard configuration" },
{ 0x1EC, 0x1EC, 0, "Interrupt Status Register" }, { 0x1ED, 0x1ED, 0, "Status LED" }, { 0x1EE,
0x1EF, 0, "Watchdog Timer Control" }, { 0x1F0, 0x1FF, 0, "IDE Controller #1" }, { 0x200,
0x277, 1, "Free" }, { 0x278, 0x27F, 1, "Free (Option for LPT)" }, { 0x280, 0x2A7, 1, "Free" },
{ 0x2A8, 0x2AF, 1, "Free (Option for on, Oxbard serial ports)" }, { 0x2B0, 0x2DF, 0, "Video
Controllers" }, { 0x2E0, 0x2E7, 1, "Free" }, { 0x2E8, 0x2EF, 0, "COM4 ? (Default)" }, { 0x2F0,
0x2F7, 1, "Free" }, { 0x2F8, 0x2FF, 0, "COM2 ? (Default)" }, { 0x300, 0x377, 1, "Free" },
{ 0x378, 0x37B, 0, "LPT (Default)" }, { 0x37C, 0x3A7, 1, "Free" }, { 0x3A8, 0x3AF, 1, "Free
(Option for on, Oxbard serial ports)" }, { 0x3B0, 0x3BB, 0, "Video Controllers" }, { 0x3BC,
0x3BF, 1, "Free (Option for LPT)" }, { 0x3C0, 0x3DF, 0, "Video Controllers" }, { 0x3E0, 0x3E7,
1, "Free" }, { 0x3E8, 0x3EF, 0, "COM3 ? (Default)" }, { 0x3F0, 0x3F7, 1, "Free" }, { 0x000,
0, NULL } };
```

File

idi.c ([see page 373](#))

Description

This is variable lpm_lx800_port_list.

3.2.4.43 svn_revision_string Variable

C++

```
const char svn_revision_string[] = { SVN_REV };
```

File

idi.c ([see page 373](#))

Description

Software revision as determined by subversion commits

3.2.5 Macros

The following table lists macros in this documentation.

Macros

Name	Description
<code>__STOPWATCH_H__</code> (see page 340)	This is macro __STOPWATCH_H__.
<code>AS_AMS1259_REGISTER_QTY</code> (see page 340)	brief Store parameters in this data structure and then can easily read/write them.
<code>BANK_EXTRACT_DEFINITION</code> (see page 340)	This is macro BANK_EXTRACT_DEFINITION.
<code>BANK_EXTRACT_ENUM</code> (see page 341)	
<code>BANK_INFO_DEFINITION</code> (see page 341)	This is macro BANK_INFO_DEFINITION.
<code>BANK_NULL_DEFINITION</code> (see page 341)	This is macro BANK_NULL_DEFINITION.
<code>BUFFER_COLUMN</code> (see page 342)	This is macro BUFFER_COLUMN.
<code>BUFFER_LENGTH</code> (see page 342)	This is macro BUFFER_LENGTH.
<code>CLOCK_PERIOD_SEC</code> (see page 342)	brief Board clock period as defined by the on-board oscillator which is 50MHz
<code>CMD_MAIN_EXTRACT_BOARD_TYPE_REQUIRED</code> (see page 342)	This is macro CMD_MAIN_EXTRACT_BOARD_TYPE_REQUIRED.
<code>CMD_MAIN_EXTRACT_COMMANDS</code> (see page 343)	This is macro CMD_MAIN_EXTRACT_COMMANDS.
<code>CMD_MAIN_LIST</code> (see page 343)	brief Used to build out the command list data structure and the board type requirement list. Implemented in this fashion in order to keep all the data together.
<code>DIN_TEST_DEBUG_PRINT</code> (see page 344)	brief

EC_EXTRACT_ENUM (see page 344)	
EC_EXTRACT_HUMAN_READABLE (see page 344)	This is macro EC_EXTRACT_HUMAN_READABLE.
EC_HUMAN_READABLE_TERMINATE (see page 345)	This is macro EC_HUMAN_READABLE_TERMINATE.
ERROR_CODES (see page 345)	brief An organized error code listing. This macro is used to build the complete error code enumeration list.
false (see page 346)	This is macro false.
FRAM_BLOCK_SIZE (see page 346)	brief
IAI_REGISTER_SET_DEFINITION (see page 346)	
IDI_BANK_INFO_DEFINITION (see page 348)	brief The bank register mapping. This mapping is upwardly compatible with the legacy hardware banking register. It also allows for future expansion utilizing the lower bits which are currently unused.
IDI_BOARD_INIT_FILE_NAME (see page 348)	brief Board type information
IDI_DIN_GROUP_QTY (see page 349)	This is macro IDI_DIN_GROUP_QTY.
IDI_DIN_GROUP_SIZE (see page 349)	
IDI_DIN_QTY (see page 349)	This is macro IDI_DIN_QTY.
IDI_DIN_SHIFT_RIGHT (see page 349)	This is macro IDI_DIN_SHIFT_RIGHT.
IDI_IO_DIRECTION_TEST (see page 350)	brief Produces an I/O direction error report.
IDI_REGISTER_SET_DEFINITION (see page 350)	brief Organized list of registers and associate attributes of each of the registers. This macro does not consume any memory of its own, and is only 'consumed' or used to automatically build enumerations and pre-built data structures.
IDO_BANK_INFO_DEFINITION (see page 352)	This is macro IDO_BANK_INFO_DEFINITION.
IDO_BOARD_INIT_FILE_NAME (see page 353)	This is macro IDO_BOARD_INIT_FILE_NAME.
IDO_DO_GROUP_QTY (see page 353)	This is macro IDO_DO_GROUP_QTY.
IDO_DO_GROUP_SIZE (see page 353)	
IDO_DO_QTY (see page 353)	This is macro IDO_DO_QTY.
IDO_DO_SHIFT_RIGHT (see page 354)	This is macro IDO_DO_SHIFT_RIGHT.
IDO_REGISTER_SET_DEFINITION (see page 354)	This is macro IDO_REGISTER_SET_DEFINITION.
INT32_MAX (see page 355)	This is macro INT32_MAX.
INT32_MIN (see page 355)	This is macro INT32_MIN.
INTERRUPT (see page 356)	This is macro INTERRUPT.
IO_TRACE_DEFINITION (see page 356)	Trace triggers
IO_TRACE_DUMP_FILE_NAME (see page 356)	This is macro IO_TRACE_DUMP_FILE_NAME.
IO_TRACE_EXTRACT_ENUM (see page 357)	This is macro IO_TRACE_EXTRACT_ENUM.
IO_TRACE_EXTRACT_NAME (see page 357)	This is macro IO_TRACE_EXTRACT_NAME.
IO_TRACE_FILE_NAME_SIZE (see page 357)	This is macro IO_TRACE_FILE_NAME_SIZE.
IO_TRACE_SIZE (see page 357)	
IO_TRACE_START_DEFINITION (see page 358)	brief Refer to IO_Trace_Log_Dataset (see page 212)() function for action details. We put all this in the x-macro as a convenient summary.
IO_TRACE_START_STOP_EXTRACT_ENUM (see page 358)	
IO_TRACE_START_STOP_EXTRACT_NAME (see page 358)	This is macro IO_TRACE_START_STOP_EXTRACT_NAME.
IO_TRACE_START_STOP_EXTRACT_NAME_HELP_LIST (see page 359)	

IO_TRACE_START_STOP_EXTRACT_SWITCH_STATEMENT (see page 359)	This is macro IO_TRACE_START_STOP_EXTRACT_SWITCH_STATEMENT.
IO_TRACE_STOP_DEFINITION (see page 359)	This is macro IO_TRACE_STOP_DEFINITION.
IO_TRACE_USE_AS_LOGGING (see page 360)	This is macro IO_TRACE_USE_AS_LOGGING.
IO_TRACE_USE_FRAM_LOGGING (see page 360)	This is macro IO_TRACE_USE_FRAM_LOGGING.
IO_TRACE_USE_SPI_LOGGING (see page 360)	
IOKERN_DOS_INT_GET (see page 360)	
IOKERN_DOS_INT_SET (see page 361)	This is macro IOKERN_DOS_INT_SET.
IOKERN_DOS_NSEOI (see page 361)	End Of Interrupt
IOKERN_DOS_PIC1_BASE_ADDRESS (see page 361)	These are the port addresses of the 8259 Programmable Interrupt Controller (PIC).
IOKERN_DOS_PIC1_CMD (see page 362)	PIC1 Command Port
IOKERN_DOS_PIC1_IMR (see page 362)	PIC1 interrupt mask port
IOKERN_DOS_PIC2_BASE_ADDRESS (see page 362)	This is macro IOKERN_DOS_PIC2_BASE_ADDRESS.
IOKERN_DOS_PIC2_CMD (see page 362)	PIC2 Command Port
IOKERN_DOS_PIC2_IMR (see page 363)	PIC2 interrupt mask port
IOKERN_IRQ_CHAIN_SELECT (see page 363)	This is macro IOKERN_IRQ_CHAIN_SELECT.
IOKERN_IRQ_ENABLE (see page 364)	This is macro IOKERN_IRQ_ENABLE.
IOKERN_IRQ_END (see page 364)	This is macro IOKERN_IRQ_END.
IOKERN_IRQ_START (see page 364)	This is macro IOKERN_IRQ_START.
IOKERN_LOCAL_IRQ_RESTORE (see page 364)	This is macro IOKERN_LOCAL_IRQ_RESTORE.
IOKERN_LOCAL_IRQ_SAVE (see page 365)	
IOKERN_TASK_QTY (see page 365)	This is macro IOKERN_TASK_QTY.
MESSAGE_SIZE (see page 366)	This is macro MESSAGE_SIZE.
REG_EXTRACT_DEFINITION (see page 366)	brief Use to build the register definitions of the IDI and IDO boards.
REG_EXTRACT_ENUM (see page 366)	brief Use to build the register enumerations of the IDI and IDO boards.
REG_LOCATION_CHANNEL_GET (see page 367)	This is macro REG_LOCATION_CHANNEL_GET.
REG_LOCATION_LOGICAL_GET (see page 367)	
REG_LOCATION_SET (see page 367)	brief 'Index' is the row number, in this case it is the 'logical address' column in the register definition x-macros. The 'channel' is an index for channel information which in this case we are setting to zero as it is not required since that information is already embodied within the logical address for the specific register definition.
REGS_INIT_QTY (see page 367)	This is macro REGS_INIT_QTY.
SPI_BLOCK_SIZE (see page 368)	This is macro SPI_BLOCK_SIZE.
SPI_COMMIT_BIT_POSITION (see page 368)	brief
SPI_REGISTER_SET_DEFINITION (see page 368)	This is macro SPI_REGISTER_SET_DEFINITION.
STOPWATCH_ERROR (see page 369)	This is macro STOPWATCH_ERROR.
STOPWATCH_HANDLE_TYPE (see page 369)	This is macro STOPWATCH_HANDLE_TYPE.
STOPWATCH_INACTIVE (see page 370)	This is macro STOPWATCH_INACTIVE.
STOPWATCH_PAUSE (see page 370)	This is macro STOPWATCH_PAUSE.
STOPWATCH_QUANTITY (see page 370)	This is macro STOPWATCH_QUANTITY.

STOPWATCH_RESET (see page 370)	This is macro STOPWATCH_RESET.
STOPWATCH_START (see page 371)	This is macro STOPWATCH_START.
STOPWATCH_STATE_TYPE (see page 371)	This is macro STOPWATCH_STATE_TYPE.
strcmpl (see page 371)	kbhit() and getch()
SVN_REV (see page 372)	brief The Subversion (SVN) time/date marker which is updated during commit of the source file to the repository.
TARGET_CPU_INTEL_386 (see page 372)	UINT_MAX
true (see page 372)	This is macro true.
UINT32_MAX (see page 372)	This is macro UINT32_MAX.

3.2.5.1 __STOPWATCH_H__ Macro

C++

```
#define __STOPWATCH_H__
```

File

stopwatch.h (see page 580)

Description

This is macro __STOPWATCH_H__.

3.2.5.2 AS_ADS1259_REGISTER_QTY Macro

C++

```
#define AS_ADS1259_REGISTER_QTY 9
```

File

idi.c (see page 373)

Description

brief Store parameters in this data structure and then can easily read/write them.

3.2.5.3 BANK_EXTRACT_DEFINITION Macro

C++

```
#define BANK_EXTRACT_DEFINITION(symbol,value) { symbol, #symbol },
```

File

idi.c (see page 373)

Description

This is macro BANK_EXTRACT_DEFINITION.

3.2.5.4 BANK_EXTRACT_ENUM Macro

C++

```
#define BANK_EXTRACT_ENUM(symbol,value) symbol = value,
```

File

idi.c (see page 373)

3.2.5.5 BANK_INFO_DEFINITION Macro

C++

```
#define BANK_INFO_DEFINITION(_)\n    _(_BANK_0,           0x00      ) \\ \n    _(_BANK_1,           0x40      ) \\ \n    _(_BANK_2,           0x80      ) \\ \n    _(_BANK_3,           0xC0      ) \\ \n    _(_BANK_4,           0x20      ) \\ \n    _(_BANK_5,           0x60      ) \\ \n    _(_BANK_6,           0xA0      ) \\ \n    _(_BANK_7,           0xE0      ) \\ \n    _(_BANK_NONE,        0xFE      ) \\ \n    _(_BANK_UNDEFINED,   0xFF      ) \\
```

File

idi.c (see page 373)

Description

This is macro BANK_INFO_DEFINITION.

3.2.5.6 BANK_NULL_DEFINITION Macro

C++

```
#define BANK_NULL_DEFINITION { BANK_UNDEFINED,     NULL }
```

File

idi.c (see page 373)

Description

This is macro BANK_NULL_DEFINITION.

3.2.5.7 BUFFER_COLUMN Macro

C++

```
#define BUFFER_COLUMN 2
```

File

idi.c (see page 373)

Description

This is macro BUFFER_COLUMN.

3.2.5.8 BUFFER_LENGTH Macro

C++

```
#define BUFFER_LENGTH 2048
```

File

idi.c (see page 373)

Description

This is macro BUFFER_LENGTH.

3.2.5.9 CLOCK_PERIOD_SEC Macro

C++

```
#define CLOCK_PERIOD_SEC 20.0e-9
```

File

idi.c (see page 373)

Description

brief Board clock period as defined by the on-board oscillator which is 50MHz

3.2.5.10 CMD_MAIN_EXTRACT_BOARD_TYPE_REQUIRED Macro

C++

```
#define CMD_MAIN_EXTRACT_BOARD_TYPE_REQUIRED(link,fnc,name,help,brdrqrd) brdrqrd,
```

File

idi.c (see page 373)

Description

This is macro CMD_MAIN_EXTRACT_BOARD_TYPE_REQUIRED.

3.2.5.11 CMD_MAIN_EXTRACT_COMMANDS Macro

C++

```
#define CMD_MAIN_EXTRACT_COMMANDS(link,fnc,name,help,brdrqrd) { link, fnc, name, help },
```

File

idi.c (see page 373)

Description

This is macro CMD_MAIN_EXTRACT_COMMANDS.

3.2.5.12 CMD_MAIN_LIST Macro

C++

```
#define CMD_MAIN_LIST(_)\n    /* link    function      name  help description\n     _( NULL,           Command_Line_Dump,          /* board type required? */\n        CSV format,       true ) \\ \"dump\",           \"Register information in\n     _( cmd_set,        Command_Line_Set,          \"set\",            \"Set/Get main\n     parameters,       true ) \\ \"spi\",           \"SPI related\n     _( cmd_spi,        Command_Line_SPI,          \"as\",             \"Analog Stick related\n     functions,        true ) \\ \"fram\",           \"FRAM related\n     _( cmd_as,         Command_Line_Analog_Stick,\n     functions,        true ) \\ \"di\",              \"Digital input related\n     _( cmd_fram,       Command_Line_FRAM,          \"do\",              \"Digital output related\n     functions,        true ) \\ \"o\",                \"unrestricted I/O write\n     _( cmd_idi48,      Command_Line_IDI48,\n     functions,        false /* implied board ID */ ) \\ \"i\",                \"unrestricted I/O read\n     _( cmd_ido48,      Command_Line_IDO48,\n     functions,        false /* implied board ID */ ) \\ \"wait\",            \"wait for any key\n     _( NULL,           Command_Line_Wait,\n     press\",           false ) \\ \"help\",            \"very brief command\n     _( NULL,           Command_Line_Help,\n     summary\",          false ) \\ \"fpga\",            \"fpga compilation and svn\n     _( NULL,           Command_Line_FPGA,\n     dates\",            false ) \\ NULL,\n     _( NULL,           NULL,\n     NULL,             false )
```

File

idi.c (see page 373)

Description

brief Used to build out the command list data structure and the board type requirement list. Implemented in this fashion in order to keep all the data together.

3.2.5.13 DIN_TEST_DEBUG_PRINT Macro

C++

```
#define DIN_TEST_DEBUG_PRINT 1
```

File

idi.c (see page 373)

Returns

A zero (SUCCESS) is returned if successful, otherwise a negative error code is returned.

Description

brief

3.2.5.14 EC_EXTRACT_ENUM Macro

C++

```
#define EC_EXTRACT_ENUM(symbol,code,message) symbol = code,
```

File

idi.c (see page 373)

3.2.5.15 EC_EXTRACT_HUMAN_READABLE Macro

C++

```
#define EC_EXTRACT_HUMAN_READABLE(symbol,code,message) { code, message },
```

File

idi.c (see page 373)

Description

This is macro EC_EXTRACT_HUMAN_READABLE.

3.2.5.16 EC_HUMAN_READABLE_TERMINATE Macro

C++

```
#define EC_HUMAN_READABLE_TERMINATE { 0, NULL }
```

File

idi.c (see page 373)

Description

This is macro EC_HUMAN_READABLE_TERMINATE.

3.2.5.17 ERROR_CODES Macro

C++

```
#define ERROR_CODES(_)
/*      enum_symbol           code  human readable      */ \
_( SUCCESS,                      0,      "" ) \
_( EC_BUFFER_TOO_LARGE,          1,      "buffer too large" ) \
_( EC_DIRECTION,                 2,      "direction" ) \
_( EC_PARAMETER,                 3,      "parameter" ) \
_( EC_NOT_FOUND,                 4,      "not found" ) \
_( EC_PARAMETER_MISSING,         5,      "missing parameter" ) \
_( EC_SYNTAX,                    6,      "syntax error" ) \
_( EC_HEX_DUMP_COUNT,           7,      "hex dump count" ) \
_( EC_INIT_FILE,                 8,      "write init file failed" ) \
_( EC_BANK,                      9,      "bank not found" ) \
/* EC_BUSY emulates a similar Linux error */
_( EC_APP_TERMINATE,            10,     "terminated by user" ) \
_( EC_APP_TERMINATE_CTRL_C,      11,     "terminated by ctrl-c" ) \
_( EC_NOT_IMPLEMENTED,           12,     "function not implemented" ) \
_( EC_CHANNEL,                  13,     "channel range" ) \
_( EC_MISSING_DATA,              14,     "missing data" ) \
_( EC_BUSY,                      16,     "interrupt busy" ) \
_( EC_INTERRUPT_UNAVAILABLE,     21,     "interrupt not available" ) \
/* was EC_INTR_ERROR was EC_EINVAL to emulate Linux behavior */
_( EC_INTR_ERROR,                22,     "interrupt error" ) \
_( EC_SPI_ECD_OUT_OF_RANGE,      40,     "SPI ECD range" ) \
_( EC_SPI_HALF_CLOCK_OUT_OF_RANGE, 41,     "SPI half clock range" ) \
_( EC_SPI_CS_B_OUT_OF_RANGE,     42,     "SPI CSB range" ) \
_( EC_SPI_NOT_FOUND,             43,     "SPI not found" ) \
_( EC_SPI_BUFFER_SIZE_ODD,       44,     "buffer size odd" ) \
_( EC_SPI_BUFFER_SIZE,           45,     "buffer size out of range" ) \
_( EC_SPI_OBJECT_SIZE,           46,     "spi tx/rx object size" ) \
_( EC_SPI_BUFFER_MISMATCH,       47,     "spi buffer mismatch" ) \
_( EC_SPI_BUFFERS_NULL,          48,     "spi tx/rx buffers null" ) \
_( EC_FRAM_ADDRESS_RANGE,        49,     "fram address range" ) \
_( EC_TEST_FAIL,                 50,     "test fail" ) \
_( EC_BOARD,                     51,     "board invalid" ) \
_( EC_BOARD_TYPE,                52,     "board type invalid" ) \
_( EC_REGISTER_ADDRESS_EXCEEDED, 53,     "register address exceeded" ) \
_( EC_EXTRA_IGNORED,             54,     "extra ignored" ) \
_( EC_FILE_ERROR,                55,     "file open error" ) \
_( EC_FILE_ERROR_SIZE,            56,     "file data size error" ) \
```

```

_( EC_MODE_LEGACY,
_( EC_BUFFER_EXCEEDED,
_( EC_RANGE,
57,      "S/W set to legacy mode"      ) \
58,      "buffer exceeded"           ) \
59,      "range exceeded"           )

```

File

idi.c (see page 373)

Description

brief An organized error code listing. This macro is used to build the complete error code enumeration list.

3.2.5.18 false Macro

C++

```
#define false 0
```

File

idi.c (see page 373)

Description

This is macro false.

3.2.5.19 FRAM_BLOCK_SIZE Macro

C++

```
#define FRAM_BLOCK_SIZE 256
```

File

idi.c (see page 373)

Description

brief

3.2.5.20 IAI_REGISTER_SET_DEFINITION Macro

C++

```

#define IAI_REGISTER_SET_DEFINITION(_)
/*   enum_symbol          logical    physical   byte   bus
byte
/*   enum_symbol          address     offset     width   aperture   read/write
acronym   bank          */ \
/** DATA REGISTERS ***/ \
_( IAI_AIN0,           0,        0x00,       2,        1,        REG_DIR_READ,
"ain",                IAI_BANK_NONE ) \

```

"ai2",	_ (IAI_AIN1, IAI_BANK_NONE)	1,) \ 2,	0x02, 0x04,)	2, 2,	1, 1,	REG_DIR_READ, REG_DIR_READ,
"ai3",	_ (IAI_AIN2, IAI_BANK_NONE)	3,) \ 4,	0x06, 0x20,)	2, 2,	1, 1,	REG_DIR_READ, REG_DIR_READ,
"ai4",	_ (IAI_AIN3, IAI_BANK_NONE)	5,) \ 6,	0x22, 0x24,)	2, 2,	1, 1,	REG_DIR_READ, REG_DIR_READ,
"ai5",	_ (IAI_AIN4, IAI_BANK_NONE)	7,) \ 8,	0x26, 0x40,)	2, 2,	1, 1,	REG_DIR_READ, REG_DIR_READ,
"ai6",	_ (IAI_AIN5, IAI_BANK_NONE)	9,) \ 10,	0x42, 0x44,)	2, 2,	1, 1,	REG_DIR_READ, REG_DIR_READ,
"ai7",	_ (IAI_AIN6, IAI_BANK_NONE)	11,) \ 12,	0x46, 0x60,)	2, 2,	1, 1,	REG_DIR_READ, REG_DIR_READ,
"ai8",	_ (IAI_AIN7, IAI_BANK_NONE)	13,) \ 14,	0x62, 0x64,)	2, 2,	1, 1,	REG_DIR_READ, REG_DIR_READ,
"ai9",	_ (IAI_AIN8, IAI_BANK_NONE)	15,) \ 16,	0x66,)	2,)	1,)	REG_DIR_READ, REG_DIR_READ,
"ai10",	_ (IAI_AIN9, IAI_BANK_NONE)	17,) \ 18,	0x44, 0x46,)	2, 2,	1, 1,	REG_DIR_READ, REG_DIR_READ,
"ai11",	_ (IAI_AIN10, IAI_BANK_NONE)	19,) \ 20,	0x46, 0x6E,)	2, 1,	1, 1,	REG_DIR_READ, REG_DIR_READ,
"ai12",	_ (IAI_AIN11, IAI_BANK_NONE)	21,) \ 22,	0x60, 0x6F,)	2, 1,	1, 1,	REG_DIR_READ, REG_DIR_READ,
"ai13",	_ (IAI_AIN12, IAI_BANK_NONE)	23,) \ 24,	0x62, 0x73,)	2, 1,	1, 1,	REG_DIR_READ, REG_DIR_READ,
"ai14",	_ (IAI_AIN13, IAI_BANK_NONE)	25,) \ 26,	0x64, 0x72,)	2, 1,	1, 1,	REG_DIR_READ, REG_DIR_READ,
"ai15",	_ (IAI_AIN14, IAI_BANK_NONE)	27,) \ 28,	0x66, 0x74,)	2, 1,	1, 1,	REG_DIR_READ, REG_DIR_READ,
"ai16",	_ (IAI_AIN15, IAI_BANK_NONE)	29,) \ 30,	0x70, 0x75,)	1, 1,	1, 1,	REG_DIR_READ, REG_DIR_READ,
<i>/* FPGA INFORMATION */\</i>						
"fd",	_ (IAI_FPGA_DATA, IDI_BANK_0)	31,) \ 32,	0x76, 0x77,)	1, 1,	1, 1,	REG_DIR_READ, REG_DIR_READ,
REG_DIR_READ_WRITE, "fi", <i>/* OTHER REGISTERS */\</i>						
"idlsb",	_ (IAI_ID_LSB, IAI_BANK_NONE)	33,) \ 34,	0x78, 0x79,)	1, 1,	1, 1,	REG_DIR_READ, REG_DIR_READ,
"idmsb",	_ (IAI_ID_MSB, IAI_BANK_NONE)	35,) \ 36,	0x7A, 0x7B,)	1, 1,	1, 1,	REG_DIR_READ, REG_DIR_READ,
<i>/* SPI REGISTERS */\</i>						
REG_DIR_READ,	_ (IAI_SPI_ID_LSB, sidlsb, IAI_BANK_NONE)	37,) \ 38,	0x7C, 0x7D,)	1, 1,	1, 1,	REG_DIR_READ, REG_DIR_READ,
REG_DIR_READ,	_ (IAI_SPI_ID_MSB, sidmsb, IAI_BANK_NONE)	39,) \ 40,	0x7E, 0x7F,)	1, 1,	1, 1,	REG_DIR_READ, REG_DIR_READ,
REG_DIR_READ_WRITE,	_ (IAI_SPI_CONFIG, scfg, IAI_BANK_NONE)	41,) \ 42,	0x80, 0x81,)	1, 1,	1, 1,	REG_DIR_READ, REG_DIR_READ,
REG_DIR_READ_WRITE,	_ (IAI_SPI_ECD, secd, IAI_BANK_NONE)	43,) \ 44,	0x82, 0x83,)	1, 1,	1, 1,	REG_DIR_READ, REG_DIR_READ,
REG_DIR_READ_WRITE,	_ (IAI_SPI_HCI_LSB, shclsb, IAI_BANK_NONE)	45,) \ 46,	0x84, 0x85,)	1, 1,	1, 1,	REG_DIR_READ, REG_DIR_READ,
REG_DIR_READ_WRITE,	_ (IAI_SPI_HCI_MSB, shcmsb, IAI_BANK_NONE)	47,) \ 48,	0x86, 0x87,)	1, 1,	1, 1,	REG_DIR_READ, REG_DIR_READ,
REG_DIR_READ_WRITE,	_ (IAI_SPI_DATA, sdata, IAI_BANK_NONE)	49,) \ 50,	0x88, 0x89,)	1, 1,	1, 1,	REG_DIR_READ, REG_DIR_READ,
"stxf",	_ (IAI_SPI_TX_STATUS, IAI_BANK_NONE)	51,) \ 52,	0x8A, 0x8B,)	1, 1,	1, 1,	REG_DIR_READ, REG_DIR_READ,
"srxf",	_ (IAI_SPI_RX_STATUS, IAI_BANK_NONE)	53,) \ 54,	0x8C, 0x8D,)	1, 1,	1, 1,	REG_DIR_READ, REG_DIR_READ,
REG_DIR_READ_WRITE,	_ (IAI_SPI_COMMIT, scmt, IAI_BANK_NONE)	55,) \ 56,	0x8E, 0x8F,)	1, 1,	1, 1,	REG_DIR_READ, REG_DIR_READ,
"",	_ (IAI_UNDEFINED, IAI_BANK_UNDEFINED)	57,) \ 58,	0x90, 0x91,)	0, 0,	0, 0,	REG_DIR_NONE, REG_DIR_NONE,

File

idi.c (see page 373)

Todo

incomplete IAI16 register set definition below

3.2.5.21 IDI_BANK_INFO_DEFINITION Macro

C++

```
#define IDI_BANK_INFO_DEFINITION(_)\n    _(_(IDI_BANK_0,           0x00) ) \\ \n    _(_(IDI_BANK_1,           0x40) ) \\ \n    _(_(IDI_BANK_2,           0x80) ) \\ \n    _(_(IDI_BANK_3,           0xC0) ) \\ \n    _(_(IDI_BANK_4,           0x20) ) \\ \n    _(_(IDI_BANK_5,           0x60) ) \\ \n    _(_(IDI_BANK_6,           0xA0) ) \\ \n    _(_(IDI_BANK_7,           0xE0) ) \\ \n    _(_(IDI_BANK_NONE,        0xFE) ) \\ \n    _(_(IDI_BANK_UNDEFINED,   0xFF) )
```

File

idi.c (see page 373)

Description

brief The bank register mapping. This mapping is upwardly compatible with the legacy hardware banking register. It also allows for future expansion utilizing the lower bits which are currently unused.

3.2.5.22 IDI_BOARD_INIT_FILE_NAME Macro

C++

```
#define IDI_BOARD_INIT_FILE_NAME "idi_init.bin"
```

File

idi.c (see page 373)

Description

brief Board type information

3.2.5.23 IDI_DIN_GROUP_QTY Macro

C++

```
#define IDI_DIN_GROUP_QTY 6
```

File

idi.c (see page 373)

Description

This is macro IDI_DIN_GROUP_QTY.

3.2.5.24 IDI_DIN_GROUP_SIZE Macro

C++

```
#define IDI_DIN_GROUP_SIZE 8
```

File

idi.c (see page 373)

3.2.5.25 IDI_DIN_QTY Macro

C++

```
#define IDI_DIN_QTY ( IDI_DIN_GROUP_SIZE * IDI_DIN_GROUP_QTY )
```

File

idi.c (see page 373)

Description

This is macro IDI_DIN_QTY.

3.2.5.26 IDI_DIN_SHIFT_RIGHT Macro

C++

```
#define IDI_DIN_SHIFT_RIGHT 3
```

File

idi.c (see page 373)

Description

This is macro IDI_DIN_SHIFT_RIGHT.

3.2.5.27 IDI_IO_DIRECTION_TEST Macro

C++

```
#define IDI_IO_DIRECTION_TEST 1
```

File

idi.c (see page 373)

Description

brief Produces an I/O direction error report.

3.2.5.28 IDI_REGISTER_SET_DEFINITION Macro

C++

```
#define IDI_REGISTER_SET_DEFINITION(_)
/*      enum_symbol          logical    physical   byte   bus
byte
/*      enum_symbol          address    offset     width   aperture   read/write
acronym bank
/** DATA REGISTERS **/ \
_( IDI_DI_GROUP0,           0,        0,         1,       1,       REG_DIR_READ,
"dig0",                   IDI_BANK_NONE ) \
_( IDI_DI_GROUP1,           1,        1,         1,       1,       REG_DIR_READ,
"dig1",                   IDI_BANK_NONE ) \
_( IDI_DI_GROUP2,           2,        2,         1,       1,       REG_DIR_READ,
"dig2",                   IDI_BANK_NONE ) \
_( IDI_DI_GROUP3,           3,        3,         1,       1,       REG_DIR_READ,
"dig3",                   IDI_BANK_NONE ) \
_( IDI_DI_GROUP4,           4,        4,         1,       1,       REG_DIR_READ,
"dig4",                   IDI_BANK_NONE ) \
_( IDI_DI_GROUP5,           5,        5,         1,       1,       REG_DIR_READ,
"dig5",                   IDI_BANK_NONE ) \
/** STATUS REGISTER **/ \
_( IDI_INTR_BY_GROUP,       6,        6,         1,       1,       REG_DIR_READ,
"isbg",                   IDI_BANK_NONE ) \
/** DATA / CONTROL REGISTERS **/ \
_( IDI_PEND_GROUP0,         7,        8,         1,       1,       REG_DIR_READ,
"REG_DIR_READ",            "p0",     IDI_BANK_3 ) \
_( IDI_PEND_GROUP1,         8,        9,         1,       1,       REG_DIR_READ,
"REG_DIR_READ",            "p1",     IDI_BANK_3 ) \
_( IDI_PEND_GROUP2,         9,        10,        1,       1,       REG_DIR_READ,
"REG_DIR_READ",             "p2",    IDI_BANK_3 ) \
_( IDI_PEND_GROUP3,         10,       11,        1,       1,       REG_DIR_READ,
"REG_DIR_READ",             "p3",    IDI_BANK_3 ) \
_( IDI_PEND_GROUP4,         11,       12,        1,       1,       REG_DIR_READ,
"REG_DIR_READ",             "p4",    IDI_BANK_3 ) \
_( IDI_PEND_GROUP5,         12,       13,        1,       1,       REG_DIR_READ,
"REG_DIR_READ",             "p5",    IDI_BANK_3 )
```

```

    _( IDI_CLEAR_GROUP0,
      IDI_BANK_3 ) \
    _( IDI_CLEAR_GROUP1,
      IDI_BANK_3 ) \
    _( IDI_CLEAR_GROUP2,
      IDI_BANK_3 ) \
    _( IDI_CLEAR_GROUP3,
      IDI_BANK_3 ) \
    _( IDI_CLEAR_GROUP4,
      IDI_BANK_3 ) \
    _( IDI_CLEAR_GROUP5,
      IDI_BANK_3 ) \
    /* CONTROL REGISTERS */ \
    _( IDI_BANK,
      IDI_BANK_NONE ) \
    REG_DIR_READ_WRITE, "bank",
    13, 8, 1, 1, REG_DIR_WRITE,
    14, 9, 1, 1, REG_DIR_WRITE,
    15, 10, 1, 1, REG_DIR_WRITE,
    16, 11, 1, 1, REG_DIR_WRITE,
    17, 12, 1, 1, REG_DIR_WRITE,
    18, 13, 1, 1, REG_DIR_WRITE,
    19, 7, 1, 1, REG_DIR_WRITE,
    20, 14, 1, 1, REG_DIR_READ,
    21, 15, 1, 1, REG_DIR_READ,
    22, 8, 1, 1, REG_DIR_READ,
    23, 9, 1, 1, REG_DIR_READ,
    24, 10, 1, 1, REG_DIR_READ,
    25, 11, 1, 1, REG_DIR_READ,
    26, 12, 1, 1, REG_DIR_READ,
    27, 13, 1, 1, REG_DIR_READ,
    28, 8, 1, 1, REG_DIR_READ,
    29, 9, 1, 1, REG_DIR_READ,
    REG_DIR_READ_WRITE, "fi",
    /* CONFIG REGISTERS */ \
    _( IDI_EDGE_GROUP0,
      IDI_BANK_1 ) \
    30, 8, 1, 1, REG_DIR_WRITE,
    31, 9, 1, 1, REG_DIR_WRITE,
    32, 10, 1, 1, REG_DIR_WRITE,
    33, 11, 1, 1, REG_DIR_WRITE,
    34, 12, 1, 1, REG_DIR_WRITE,
    35, 13, 1, 1, REG_DIR_WRITE,
    36, 8, 1, 1, REG_DIR_WRITE,
    37, 9, 1, 1, REG_DIR_WRITE,
    38, 10, 1, 1, REG_DIR_WRITE,
    39, 11, 1, 1, REG_DIR_WRITE,
    40, 12, 1, 1, REG_DIR_WRITE,
    41, 13, 1, 1, REG_DIR_WRITE,
    /* SPI REGISTERS */ \
    42, 8, 1, 1, REG_DIR_WRITE,
    REG_DIR_READ, "sidlsb",
    IDI_BANK_6 ) \

```

```

_( IDI_SPI_ID_MSB,           43,      9,      1,      1,
REG_DIR_READ, "sidmsb",     IDI_BANK_6 ) \
_( IDI_SPI_CONFIG,          44,      10,     1,      1,
REG_DIR_READ_WRITE, "scfg", IDI_BANK_6 ) \
_( IDI_SPI_ECD,             45,      11,     1,      1,
REG_DIR_READ_WRITE, "secd", IDI_BANK_6 ) \
_( IDI_SPI_HCI_LSB,         46,      12,     1,      1,
REG_DIR_READ_WRITE, "shclsb", IDI_BANK_6 ) \
_( IDI_SPI_HCI_MSB,         47,      13,     1,      1,
REG_DIR_READ_WRITE, "shcmsb", IDI_BANK_6 ) \
_( IDI_SPI_DATA,            48,      8,      1,      1,
REG_DIR_READ_WRITE, "sdata", IDI_BANK_7 ) \
_( IDI_SPI_TX_STATUS,       49,      9,      1,      1,      REG_DIR_READ,
"stxf", IDI_BANK_7 ) \
_( IDI_SPI_RX_STATUS,       50,      10,     1,      1,      REG_DIR_READ,
"srx", IDI_BANK_7 ) \
_( IDI_SPI_COMMIT,          51,      11,     1,      1,
REG_DIR_READ_WRITE, "scmt", IDI_BANK_7 ) \
_( IDI_UNDEFINED,            52,      0,      0,      0,      REG_DIR_NONE,
" ", IDI_BANK_UNDEFINED )

```

File

idi.c (see page 373)

Description

brief Organized list of registers and associate attributes of each of the registers. This macro does not consume any memory of itsel, and is only 'consumed' or used to automatically build enumerations and pre-built data structures.

3.2.5.29 IDO_BANK_INFO_DEFINITION Macro

C++

```

#define IDO_BANK_INFO_DEFINITION(_)
_( IDO_BANK_0,           0x00 ) \
_( IDO_BANK_1,           0x40 ) \
_( IDO_BANK_2,           0x80 ) \
_( IDO_BANK_3,           0xC0 ) \
_( IDO_BANK_4,           0x20 ) \
_( IDO_BANK_5,           0x60 ) \
_( IDO_BANK_6,           0xA0 ) \
_( IDO_BANK_7,           0xE0 ) \
_( IDO_BANK_NONE,        0xFE ) \
_( IDO_BANK_UNDEFINED,   0xFF )

```

File

idi.c (see page 373)

Description

This is macro IDO_BANK_INFO_DEFINITION.

3.2.5.30 IDO_BOARD_INIT_FILE_NAME Macro

C++

```
#define IDO_BOARD_INIT_FILE_NAME "ido_init.bin"
```

File

idi.c (see page 373)

Description

This is macro IDO_BOARD_INIT_FILE_NAME.

3.2.5.31 IDO_DO_GROUP_QTY Macro

C++

```
#define IDO_DO_GROUP_QTY 6
```

File

idi.c (see page 373)

Description

This is macro IDO_DO_GROUP_QTY.

3.2.5.32 IDO_DO_GROUP_SIZE Macro

C++

```
#define IDO_DO_GROUP_SIZE 8
```

File

idi.c (see page 373)

3.2.5.33 IDO_DO_QTY Macro

C++

```
#define IDO_DO_QTY ( IDO_DO_GROUP_SIZE * IDO_DO_GROUP_QTY )
```

File

idi.c (see page 373)

Description

This is macro IDO_DO_QTY.

3.2.5.34 IDO_DO_SHIFT_RIGHT Macro

C++

```
#define IDO_DO_SHIFT_RIGHT 3
```

File

idi.c (see page 373)

Description

This is macro IDO_DO_SHIFT_RIGHT.

3.2.5.35 IDO_REGISTER_SET_DEFINITION Macro

C++

```
#define IDO_REGISTER_SET_DEFINITION(_)
/*      enum_symbol          logical    physical   byte   bus
byte
/*      enum_symbol          address    offset     width   aperture   read/write
acronym bank
/** DATA REGISTERS ***/ \
_( IDO_DO_GROUP0,           0,        0,         1,       1,
REG_DIR_READ_WRITE, "dog0", IDO_BANK_NONE ) \
_( IDO_DO_GROUP1,           1,        1,         1,       1,
REG_DIR_READ_WRITE, "dog1", IDO_BANK_NONE ) \
_( IDO_DO_GROUP2,           2,        2,         1,       1,
REG_DIR_READ_WRITE, "dog2", IDO_BANK_NONE ) \
_( IDO_DO_GROUP3,           3,        3,         1,       1,
REG_DIR_READ_WRITE, "dog3", IDO_BANK_NONE ) \
_( IDO_DO_GROUP4,           4,        4,         1,       1,
REG_DIR_READ_WRITE, "dog4", IDO_BANK_NONE ) \
_( IDO_DO_GROUP5,           5,        5,         1,       1,
REG_DIR_READ_WRITE, "dog5", IDO_BANK_NONE ) \
/** CONTROL REGISTERS ***/ \
_( IDO_BANK,                6,        7,         1,       1,
REG_DIR_READ_WRITE, "bank", IDO_BANK_NONE ) \
/** FPGA INFORMATION ***/ \
_( IDO_FPGA_DATA,           7,        8,         1,       1,       REG_DIR_READ,
"fd", IDI_BANK_0 ) \
_( IDO_FPGA_INDEX,           8,        9,         1,       1,
REG_DIR_READ_WRITE, "fi", IDI_BANK_0 ) \
/** *** */
_( IDO_ID_LSB,               9,       14,        1,       1,       REG_DIR_READ,
"idlsb", IDO_BANK_NONE ) \
_( IDO_ID_MSB,               10,      15,        1,       1,       REG_DIR_READ,
"idmsb", IDO_BANK_NONE ) \
/** SPI REGISTERS ***/ \
_( IDO_SPI_ID_LSB,           11,       8,         1,       1,
REG_DIR_READ, "sidlsb", IDO_BANK_6 ) \
```

```

_( IDO_SPI_ID_MSB,
REG_DIR_READ,           "sidmsb",      12,      9,      1,      1,
_( IDO_SPI_CONFIG,      "scfg",        13,      10,     ) \ 1,      1,
REG_DIR_READ_WRITE,    "scd",         14,      11,     ) \ 1,      1,
_( IDO_SPI_ECD,         "secd",        15,      12,     ) \ 1,      1,
REG_DIR_READ_WRITE,    "shclsb",      16,      13,     ) \ 1,      1,
REG_DIR_READ_WRITE,    "shcmsb",      17,      8,      1,      1,
REG_DIR_READ_WRITE,    "sdata",        18,      9,      1,      1,      REG_DIR_READ,
"stxf",                IDO_BANK_7,   ) \ 1,      1,
_( IDO_SPI_RX_STATUS,  "srxf",        19,      10,     ) \ 1,      1,      REG_DIR_READ,
REG_DIR_READ_WRITE,    "scmt",        20,      11,     ) \ 1,      1,
_( IDO_SPI_COMMIT,     "scmt",        21,      0,      0,      0,      REG_DIR_NONE,
" ",                   IDO_BANK_UNDEFINED )

```

File

idi.c (see page 373)

Description

This is macro IDO_REGISTER_SET_DEFINITION.

3.2.5.36 INT32_MAX Macro

C++

```
#define INT32_MAX LONG_MAX
```

File

idi.c (see page 373)

Description

This is macro INT32_MAX.

3.2.5.37 INT32_MIN Macro

C++

```
#define INT32_MIN LONG_MIN
```

File

idi.c (see page 373)

Description

This is macro INT32_MIN.

3.2.5.38 INTERRUPT Macro

C++

```
#define INTERRUPT interrupt
```

File

idi.c (see page 373)

Description

This is macro INTERRUPT.

3.2.5.39 IO_TRACE_DEFINITION Macro

C++

```
#define IO_TRACE_DEFINITION(_)
/*          enum_symbol    value     name           */
_( IO_TRACE_NULL,      0,        "NULL"          ) \
_( IO_TRACE_READ,      1,        "IO_Read_U8"    ) \
_( IO_TRACE_WRITE,     2,        "IO_Write_U8"   ) \
_( IO_TRACE_SPI,       3,        "SPI"           ) \
_( IO_TRACE_FRAM,      4,        "FRAM"          ) \
_( IO_TRACE_AS,        5,        "Analog_Stick" ) \
_( IO_TRACE_OTHER,     6,        "Other"         ) \
```

File

idi.c (see page 373)

Description

Trace triggers

3.2.5.40 IO_TRACE_DUMP_FILE_NAME Macro

C++

```
#define IO_TRACE_DUMP_FILE_NAME "iotrace.csv"
```

File

idi.c (see page 373)

Description

This is macro IO_TRACE_DUMP_FILE_NAME.

3.2.5.41 IO_TRACE_EXTRACT_ENUM Macro

C++

```
#define IO_TRACE_EXTRACT_ENUM(symbol,index,name) symbol = index,
```

File

idi.c (see page 373)

Description

This is macro IO_TRACE_EXTRACT_ENUM.

3.2.5.42 IO_TRACE_EXTRACT_NAME Macro

C++

```
#define IO_TRACE_EXTRACT_NAME(symbol,index,name) name,
```

File

idi.c (see page 373)

Description

This is macro IO_TRACE_EXTRACT_NAME.

3.2.5.43 IO_TRACE_FILE_NAME_SIZE Macro

C++

```
#define IO_TRACE_FILE_NAME_SIZE 32
```

File

idi.c (see page 373)

Description

This is macro IO_TRACE_FILE_NAME_SIZE.

3.2.5.44 IO_TRACE_SIZE Macro

C++

```
#define IO_TRACE_SIZE 1024
```

File

idi.c (see page 373)

3.2.5.45 IO_TRACE_START_DEFINITION Macro

C++

```
#define IO_TRACE_START_DEFINITION(_)
/*          enum_symbol      value     name      IO_Trace_Log_Dataset() action
 */ \
_( IO_TRACE_START_DISABLE,      0,       "disable",    ds->trace.active =
false;
_( IO_TRACE_START_ON,         1,       "on",        ds->trace.active =
true;
_( IO_TRACE_START_ERROR,      2,       "error",     if ( SUCCESS != error_code )
ds->trace.active = true; ) \
_( IO_TRACE_START_SPI,        3,       "spi",       if ( IO_TRACE_SPI ==
action ) ds->trace.active = true; ) \
_( IO_TRACE_START_FRAM,       4,       "fram",     if ( IO_TRACE_FRAM ==
action ) ds->trace.active = true; ) \
_( IO_TRACE_START_AS,         5,       "as",        if ( IO_TRACE_AS == action
) ds->trace.active = true; )
```

File

idi.c (see page 373)

Description

brief Refer to IO_Trace_Log_Dataset (see page 212)() function for action details. We put all this in the x-macro as a convenient summary.

3.2.5.46 IO_TRACE_START_STOP_EXTRACT_ENUM Macro

C++

```
#define IO_TRACE_START_STOP_EXTRACT_ENUM(symbol,index,name,action) symbol = index,
```

File

idi.c (see page 373)

3.2.5.47 IO_TRACE_START_STOP_EXTRACT_NAME Macro

C++

```
#define IO_TRACE_START_STOP_EXTRACT_NAME(symbol,index,name,action) name,
```

File

idi.c (see page 373)

Description

This is macro IO_TRACE_START_STOP_EXTRACT_NAME.

3.2.5.48 IO_TRACE_START_STOP_EXTRACT_NAME_HELP_LIST Macro

C++

```
#define IO_TRACE_START_STOP_EXTRACT_NAME_HELP_LIST(symbol,index,name,action) name " "
```

File

idi.c (see page 373)

3.2.5.49 IO_TRACE_START_STOP_EXTRACT_SWITCH_STATEMENT Macro

C++

```
#define IO_TRACE_START_STOP_EXTRACT_SWITCH_STATEMENT(symbol,index,name,action) \
    case symbol: \
        { action } \
    break;
```

File

idi.c (see page 373)

Description

This is macro IO_TRACE_START_STOP_EXTRACT_SWITCH_STATEMENT.

3.2.5.50 IO_TRACE_STOP_DEFINITION Macro

C++

```
#define IO_TRACE_STOP_DEFINITION(_) \
/*      enum_symbol           value   name      IO_Trace_Log_Dataset() action \
 */ \
    _( IO_TRACE_STOP_NONE,          0,           "none",           ) \
    _( IO_TRACE_STOP_ERROR,         1,           "error",          if ( SUCCESS != error_code ) \
        ds->trace.active = false; \
    _( IO_TRACE_STOP_CTRLC,         2,           "ctrlc",          )
```

File

idi.c (see page 373)

Description

This is macro IO_TRACE_STOP_DEFINITION.

3.2.5.51 IO_TRACE_USE_AS_LOGGING Macro

C++

```
#define IO_TRACE_USE_AS_LOGGING 1
```

File

idi.c (see page 373)

Description

This is macro IO_TRACE_USE_AS_LOGGING.

3.2.5.52 IO_TRACE_USE_FRAM_LOGGING Macro

C++

```
#define IO_TRACE_USE_FRAM_LOGGING 1
```

File

idi.c (see page 373)

Description

This is macro IO_TRACE_USE_FRAM_LOGGING.

3.2.5.53 IO_TRACE_USE_SPI_LOGGING Macro

C++

```
#define IO_TRACE_USE_SPI_LOGGING 1
```

File

idi.c (see page 373)

3.2.5.54 IOKERN_DOS_INT_GET Macro

C++

```
#define IOKERN_DOS_INT_GET(num) IOKern_DOS_Vector_Get(num)
```

File

idi.c (see page 373)

3.2.5.55 IOKERN_DOS_INT_SET Macro

C++

```
#define IOKERN_DOS_INT_SET(num, fn) IOKern_DOS_Vector_Set(num, (IOKERN_DOS_VECTOR_TYPE)fn)
```

File

idi.c (see page 373)

Description

This is macro IOKERN_DOS_INT_SET.

3.2.5.56 IOKERN_DOS_NSEOI Macro

C++

```
#define IOKERN_DOS_NSEOI 0x20 /* End Of Interrupt */
```

File

idi.c (see page 373)

Description

End Of Interrupt

3.2.5.57 IOKERN_DOS_PIC1_BASE_ADDRESS Macro

C++

```
#define IOKERN_DOS_PIC1_BASE_ADDRESS 0x20
```

File

idi.c (see page 373)

Description

These are the port addresses of the 8259 Programmable Interrupt Controller (PIC).

3.2.5.58 IOKERN_DOS_PIC1_CMD Macro

C++

```
#define IOKERN_DOS_PIC1_CMD IOKERN_DOS_PIC1_BASE_ADDRESS           /* PIC1 Command Port */
```

File

idi.c (see page 373)

Description

PIC1 Command Port

3.2.5.59 IOKERN_DOS_PIC1_IMR Macro

C++

```
#define IOKERN_DOS_PIC1_IMR ( IOKERN_DOS_PIC1_BASE_ADDRESS + 1 )      /* PIC1 interrupt mask port */
```

File

idi.c (see page 373)

Description

PIC1 interrupt mask port

3.2.5.60 IOKERN_DOS_PIC2_BASE_ADDRESS Macro

C++

```
#define IOKERN_DOS_PIC2_BASE_ADDRESS 0xA0
```

File

idi.c (see page 373)

Description

This is macro IOKERN_DOS_PIC2_BASE_ADDRESS.

3.2.5.61 IOKERN_DOS_PIC2_CMD Macro

C++

```
#define IOKERN_DOS_PIC2_CMD IOKERN_DOS_PIC2_BASE_ADDRESS           /* PIC2 Command Port */
```

File

idi.c (see page 373)

Description

PIC2 Command Port

3.2.5.62 IOKERN_DOS_PIC2_IMR Macro

C++

```
#define IOKERN_DOS_PIC2_IMR ( IOKERN_DOS_PIC2_BASE_ADDRESS + 1 )      /* PIC2 interrupt mask
port */
```

File

idi.c (see page 373)

Description

PIC2 interrupt mask port

3.2.5.63 IOKERN_IRQ_CHAIN_SELECT Macro

C++

```
#define IOKERN_IRQ_CHAIN_SELECT(old_fp,regs) \
    _ECX = regs.eax; \
    regs.aw.h = FP_SEG((void far *)old_fp); \
    regs.aw.l = FP_OFF((void far *)old_fp); \
    _EAX = _ECX; \
    __emit__(0x5D); \
    __emit__(0x66); __emit__(0x5F); \
    __emit__(0x66); __emit__(0x5E); \
    __emit__(0x1F); \
    __emit__(0x07); \
    __emit__(0x66); __emit__(0x5A); \
    __emit__(0x66); __emit__(0x59); \
    __emit__(0x66); __emit__(0x5B); \
    __emit__(0xCB);
```

File

idi.c (see page 373)

Description

This is macro IOKERN_IRQ_CHAIN_SELECT.

3.2.5.64 IOKERN_IRQ_ENABLE Macro

C++

```
#define IOKERN_IRQ_ENABLE enable()
```

File

idi.c (see page 373)

Description

This is macro IOKERN_IRQ_ENABLE.

3.2.5.65 IOKERN_IRQ_END Macro

C++

```
#define IOKERN_IRQ_END(irq) IOKern_PIC_EOI( irq ); enable()
```

File

idi.c (see page 373)

Description

This is macro IOKERN_IRQ_END.

3.2.5.66 IOKERN_IRQ_START Macro

C++

```
#define IOKERN_IRQ_START disable()
```

File

idi.c (see page 373)

Description

This is macro IOKERN_IRQ_START.

3.2.5.67 IOKERN_LOCAL_IRQ_RESTORE Macro

C++

```
#define IOKERN_LOCAL_IRQ_RESTORE(flags) \
    do { \
        asm push bx \
        ; \
    } \

```

```

asm mov bx, [flags]      ;    \
asm push bx              ;    \
asm popf                ;    \
asm pop bx              ;    \
} while(0)

```

File

idi.c (see page 373)

Description

This is macro IOKERN_LOCAL_IRQ_RESTORE.

3.2.5.68 IOKERN_LOCAL_IRQ_SAVE Macro

C++

```

#define IOKERN_LOCAL_IRQ_SAVE(flags) \
do { \
    asm push bx      ;    \
    asm pushf       ;    \
    asm pop bx      ;    \
    asm mov [flags], bx ;    \
    asm cli         ;    \
    asm pop bx      ;    \
} while(0)

```

File

idi.c (see page 373)

Notes

Borland 5 requires semicolons - GCC will need something different

3.2.5.69 IOKERN_TASK_QTY Macro

C++

```
#define IOKERN_TASK_QTY 17
```

File

idi.c (see page 373)

Description

This is macro IOKERN_TASK_QTY.

3.2.5.70 MESSAGE_SIZE Macro

C++

```
#define MESSAGE_SIZE 256
```

File

idi.c (see page 373)

Description

This is macro MESSAGE_SIZE.

3.2.5.71 REG_EXTRACT_DEFINITION Macro

C++

```
#define  
REG_EXTRACT_DEFINITION(symbol,index,offset,register_bytes,aperture_bytes,read_write,name,bank)  
{ REG_LOCATION_SET(index,0), read_write, bank, offset, #symbol, name },
```

File

idi.c (see page 373)

Description

brief Use to build the register definitions of the IDI and IDO boards.

3.2.5.72 REG_EXTRACT_ENUM Macro

C++

```
#define  
REG_EXTRACT_ENUM(symbol,index,offset,register_bytes,aperture_bytes,read_write,name,bank)  
symbol = REG_LOCATION_SET(index,0),
```

File

idi.c (see page 373)

Description

brief Use to build the register enumerations of the IDI and IDO boards.

3.2.5.73 REG_LOCATION_CHANNEL_GET Macro

C++

```
#define REG_LOCATION_CHANNEL_GET(location) ( (location) & 0xFF )
```

File

idi.c (see page 373)

Description

This is macro REG_LOCATION_CHANNEL_GET.

3.2.5.74 REG_LOCATION_LOGICAL_GET Macro

C++

```
#define REG_LOCATION_LOGICAL_GET(location) ( ( (location) >> 8 ) & 0xFF )
```

File

idi.c (see page 373)

3.2.5.75 REG_LOCATION_SET Macro

C++

```
#define REG_LOCATION_SET(index,channel) ( ( ((index) & 0xFF) << 8 ) | ((channel) & 0xFF) )
```

File

idi.c (see page 373)

Description

brief 'Index' is the row number, in this case it is the 'logical address' column in the register definition x-macros. The 'channel' is an index for channel information which in this case we are setting to zero as it is not required since that information is already embodied within the logical address for the specific register definition.

3.2.5.76 REGS_INIT_QTY Macro

C++

```
#define REGS_INIT_QTY 16
```

File

idi.c (see page 373)

Description

This is macro REGS_INIT_QTY.

3.2.5.77 SPI_BLOCK_SIZE Macro

C++

```
#define SPI_BLOCK_SIZE 256
```

File

idi.c (see page 373)

Description

This is macro SPI_BLOCK_SIZE.

3.2.5.78 SPI_COMMIT_BIT_POSITION Macro

C++

```
#define SPI_COMMIT_BIT_POSITION 0x01
```

File

idi.c (see page 373)

Description

brief

3.2.5.79 SPI_REGISTER_SET_DEFINITION Macro

C++

```
#define SPI_REGISTER_SET_DEFINITION(_)
/*   enum_symbol           logical    physical   byte   bus
byte
/*   enum_symbol           address    offset     width   aperture   read/write
acronym   bank           */ \
/** SPI_REGISTERS ***/ \
_( SPI_ID_LSB,           7,        8,        1,      1,
REG_DIR_READ,           "sidlsb",  BANK_6,    ) \
_( SPI_ID_MSB,           8,        9,        1,      1,
REG_DIR_READ,           "sidmsb",  BANK_6,    ) \
_( SPI_CONFIG,           9,        10,       1,      1,
REG_DIR_READ_WRITE,    "scfg",    BANK_6,    ) \
_( SPI_ECD,              10,       11,       1,      1,
REG_DIR_READ_WRITE,    "secd",    BANK_6,    ) \
_( SPI_HCI_LSB,          11,       12,       1,      1,
REG_DIR_READ_WRITE,    "shclsb",  BANK_6,    ) \
```

```

_( SPI_HCI_MSB,
REG_DIR_READ_WRITE,      12,      13,      1,      1,
_( SPI_DATA,              BANK_6   13,      8,      1,      1,
REG_DIR_READ_WRITE,      "shcmsb", 13,      8,      1,      1,
_( SPI_TX_STATUS,
"stxf",                  BANK_7   14,      9,      1,      1,      REG_DIR_READ,
_( SPI_RX_STATUS,
"srxr",                  BANK_7   15,      10,     1,      1,      REG_DIR_READ,
_( SPI_COMMIT,
"scmt",                  BANK_7   16,      11,     1,      1,
REG_DIR_READ_WRITE,      "scmt",   BANK_7   17,      0,      0,      0,      REG_DIR_NONE,
_( SPI_UNDEFINED,
" ",                     BANK_UNDEFINED ) \
)

```

File

idi.c (see page 373)

Description

This is macro SPI_REGISTER_SET_DEFINITION.

3.2.5.80 STOPWATCH_ERROR Macro

C++

```
#define STOPWATCH_ERROR -1
```

File

stopwatch.h (see page 580)

Description

This is macro STOPWATCH_ERROR.

3.2.5.81 STOPWATCH_HANDLE_TYPE Macro

C++

```
#define STOPWATCH_HANDLE_TYPE char
```

File

stopwatch.h (see page 580)

Description

This is macro STOPWATCH_HANDLE_TYPE.

3.2.5.82 STOPWATCH_INACTIVE Macro

C++

```
#define STOPWATCH_INACTIVE -1
```

File

stopwatch.h (see page 580)

Description

This is macro STOPWATCH_INACTIVE.

3.2.5.83 STOPWATCH_PAUSE Macro

C++

```
#define STOPWATCH_PAUSE 2
```

File

stopwatch.h (see page 580)

Description

This is macro STOPWATCH_PAUSE.

3.2.5.84 STOPWATCH_QUANTITY Macro

C++

```
#define STOPWATCH_QUANTITY 4
```

File

stopwatch.h (see page 580)

Description

This is macro STOPWATCH_QUANTITY.

3.2.5.85 STOPWATCH_RESET Macro

C++

```
#define STOPWATCH_RESET 0
```

File

stopwatch.h (see page 580)

Description

This is macro STOPWATCH_RESET.

3.2.5.86 STOPWATCH_START Macro

C++

```
#define STOPWATCH_START 1
```

File

stopwatch.h (see page 580)

Description

This is macro STOPWATCH_START.

3.2.5.87 STOPWATCH_STATE_TYPE Macro

C++

```
#define STOPWATCH_STATE_TYPE char
```

File

stopwatch.h (see page 580)

Description

This is macro STOPWATCH_STATE_TYPE.

3.2.5.88 strcmpl Macro

C++

```
#define strcmpl strcasecmp
```

File

idi.c (see page 373)

Description

kbhit() and getch()

3.2.5.89 SVN_REV Macro

C++

```
#define SVN_REV "$Date: 2015-10-19 16:02:35 -0500 (Mon, 19 Oct 2015) $"
```

File

idi.c (see page 373)

Description

brief The Subversion (SVN) time/date marker which is updated during commit of the source file to the repository.

3.2.5.90 TARGET_CPU_INTEL_386 Macro

C++

```
#define TARGET_CPU_INTEL_386 1
```

File

idi.c (see page 373)

Description

UINT_MAX

3.2.5.91 true Macro

C++

```
#define true 1
```

File

idi.c (see page 373)

Description

This is macro true.

3.2.5.92 UINT32_MAX Macro

C++

```
#define UINT32_MAX ULONG_MAX
```

File

idi.c (see page 373)

Description

This is macro UINT32_MAX.

3.2.6 Files

The following table lists files in this documentation.

Files

Name	Description
idi.c (see page 373)	idi.c Created on: Feb 25, 2015
stopwatch.h (see page 580)	TO DO COPYRIGHT Copyright(c) 1999-2008 by Apex Embedded Systems. All rights reserved. This document is the confidential property of Apex Embedded Systems Any reproduction or dissemination is prohibited unless specifically authorized in writing.

3.2.6.1 idi.c

idi.c

Created on: Feb 25, 2015

Author

Mike

Body Source

```
1:  
7:  
8:  
9:  
12:  
13:  
14:  
19:  
20: #include <stdio.h>  
21: #include <string.h>  
22: #include <stdlib.h>  
23: #include <signal.h>  
24: #include <limits.h>  
25: #include <time.h>  
26: #include <errno.h>
```

```
27: #include <math.h>
28:
29:
30: #if defined( __MSDOS__ )
31: # include <mem.h>
32: # include <dos.h>
33: # include <conio.h>
34: #else
35: # ifndef strcmpl
36: # define strcmpl strcasecmp
37: # endif
38: # include <stdint.h>
39: #endif
40:
41:
42: #include "stopwatch.h"
43:
44:
45:
51: #define SVN_REV "$Date: 2015-10-19 16:02:35 -0500 (Mon, 19 Oct 2015) $"
52:
53:
54:
60: #if defined( __MSDOS__ )
61:     typedef unsigned char uint8_t;
62:     typedef unsigned int uint16_t;
63:     typedef unsigned long uint32_t;
64:
65:     typedef char int8_t;
66:     typedef int int16_t;
67:     typedef long int32_t;
68:
69: # define UINT32_MAX ULONG_MAX
70: # define INT32_MAX LONG_MAX
71: # define INT32_MIN LONG_MIN
72:
73: #endif
74:
79: typedef int BOOL;
80: #define false 0
81: #define true 1
82: enum { FALSE = 0, TRUE = 1 };
83:
84:
89: #define CLOCK_PERIOD_SEC 20.0e-9
90:
91:
122: enum
123: {
124:     ID_DIN    = 0x8012,
125:     ID_SPI    = 0x8013,
126:     ID_DOUT   = 0x8021,
127:     ID_IAI    = 0x8030
128: };
129:
130:
135: enum
136: {
137:     ID_IDI48  = 0,
138:     ID_IDO48  = 1,
139:     ID_IAI16  = 2,
140:     ID_ARM32  = 3,
141:     ID_BOARD_UNKNOWN = 99
142: };
```

```

143:
144:
148:
149:
150:
151:
152:
153:
154:
155:
156:
157:
164: #define ERROR_CODES(_)
165: \
166: _(_( SUCCESS, 0, "" ) \
167: _(_( EC_BUFFER_TOO_LARGE, 1, "buffer too large" ) \
168: _(_( EC_DIRECTION, 2, "direction" ) \
169: _(_( EC_PARAMETER, 3, "parameter" ) \
170: _(_( EC_NOT_FOUND, 4, "not found" ) \
171: _(_( EC_PARAMETER_MISSING, 5, "missing parameter" ) \
172: _(_( EC_SYNTAX, 6, "syntax error" ) \
173: _(_( EC_HEX_DUMP_COUNT, 7, "hex dump count" ) \
174: _(_( EC_INIT_FILE, 8, "write init file failed" ) \
175: _(_( EC_BANK, 9, "bank not found" ) \
176: \
177: _(_( EC_APP_TERMINATE, 10, "terminated by user" ) \
178: _(_( EC_APP_TERMINATE_CTRL_C, 11, "terminated by ctrl-c" ) \
179: _(_( EC_NOT_IMPLEMENTED, 12, "function not implemented" ) \
180: _(_( EC_CHANNEL, 13, "channel range" ) \
181: _(_( EC_MISSING_DATA, 14, "missing data" ) \
182: _(_( EC_BUSY, 16, "interrupt busy" ) \
183: _(_( EC_INTERRUPT_UNAVAILABLE, 21, "interrupt not available" ) \
184: \
185: _(_( EC_INTR_ERROR, 22, "interrupt error" ) \
186: _(_( EC_SPI_ECD_OUT_OF_RANGE, 40, "SPI ECD range" ) \
187: _(_( EC_SPI_HALF_CLOCK_OUT_OF_RANGE, 41, "SPI half clock range" ) \
188: _(_( EC_SPI_CSOUT_OF_RANGE, 42, "SPI CSB range" ) \
189: _(_( EC_SPI_NOT_FOUND, 43, "SPI not found" ) \
190: _(_( EC_SPI_BUFFER_SIZE_ODD, 44, "buffer size odd" ) \
191: _(_( EC_SPI_BUFFER_SIZE, 45, "buffer size out of range" ) \
192: _(_( EC_SPI_OBJECT_SIZE, 46, "spi tx/rx object size" ) \
193: _(_( EC_SPI_BUFFER_MISMATCH, 47, "spi buffer mismatch" ) \
194: _(_( EC_SPI_BUFFERS_NULL, 48, "spi tx/rx buffers null" ) \
195: _(_( EC_FRAM_ADDRESS_RANGE, 49, "fram address range" ) \
196: _(_( EC_TEST_FAIL, 50, "test fail" ) \
197: _(_( EC_BOARD, 51, "board invalid" ) \
198: _(_( EC_BOARD_TYPE, 52, "board type invalid" ) \
199: _(_( EC_REGISTER_ADDRESS_EXCEEDED, 53, "register address exceeded" ) \
200: _(_( EC_EXTRA_IGNORED, 54, "extra ignored" ) \
201: _(_( EC_FILE_ERROR, 55, "file open error" ) \
202: _(_( EC_FILE_ERROR_SIZE, 56, "file data size error" ) \
203: _(_( EC_MODE_LEGACY, 57, "S/W set to legacy mode" ) \
204: _(_( EC_BUFFER_EXCEEDED, 58, "buffer exceeded" ) \
205: _(_( EC_RANGE, 59, "range exceeded" ) \
206:
207:
208:
209: #define EC_EXTRACT_ENUM(symbol,code,message) symbol = code,
210: #define EC_EXTRACT_HUMAN_READABLE(symbol,code,message) { code, message },
211: #define EC_HUMAN_READABLE_TERMINATE { 0, NULL }
212:
213:
214:
215:

```

```
216: typedef enum
217: {
218:     ERROR_CODES( EC_EXTRACT_ENUM )
219: } EC_ENUM;
220:
221: struct ec_human_readable
222: {
223:     EC_ENUM    error_code;
224:     const char * message;
225: };
226:
227:
228:
229:
230:
231:
232:
233:
240: #define IDI_BANK_INFO_DEFINITION(_)
241:     _( IDI_BANK_0,    0x00 ) \
242:     _( IDI_BANK_1,    0x40 ) \
243:     _( IDI_BANK_2,    0x80 ) \
244:     _( IDI_BANK_3,    0xC0 ) \
245:     _( IDI_BANK_4,    0x20 ) \
246:     _( IDI_BANK_5,    0x60 ) \
247:     _( IDI_BANK_6,    0xA0 ) \
248:     _( IDI_BANK_7,    0xE0 ) \
249:     _( IDI_BANK_NONE, 0xFE ) \
250:     _( IDI_BANK_UNDEFINED, 0xFF )
251:
252:
253: #define IDO_BANK_INFO_DEFINITION(_)
254:     _( IDO_BANK_0,    0x00 ) \
255:     _( IDO_BANK_1,    0x40 ) \
256:     _( IDO_BANK_2,    0x80 ) \
257:     _( IDO_BANK_3,    0xC0 ) \
258:     _( IDO_BANK_4,    0x20 ) \
259:     _( IDO_BANK_5,    0x60 ) \
260:     _( IDO_BANK_6,    0xA0 ) \
261:     _( IDO_BANK_7,    0xE0 ) \
262:     _( IDO_BANK_NONE, 0xFE ) \
263:     _( IDO_BANK_UNDEFINED, 0xFF )
264:
265:
266: #define BANK_INFO_DEFINITION(_)
267:     _( BANK_0,        0x00 ) \
268:     _( BANK_1,        0x40 ) \
269:     _( BANK_2,        0x80 ) \
270:     _( BANK_3,        0xC0 ) \
271:     _( BANK_4,        0x20 ) \
272:     _( BANK_5,        0x60 ) \
273:     _( BANK_6,        0xA0 ) \
274:     _( BANK_7,        0xE0 ) \
275:     _( BANK_NONE,    0xFE ) \
276:     _( BANK_UNDEFINED, 0xFF )
277:
278:
279:
280:
281:
282:
283:
284: #define BANK_EXTRACT_ENUM(symbol,value)   symbol = value,
285:
```

```
286: #define BANK_EXTRACT_DEFINITION(symbol,value) { symbol,           #symbol },
287:
288:
289: enum { IDI_BANK_ENUM_QTY = 10 };
290:
291: enum { IDO_BANK_ENUM_QTY = 10 };
292:
293: typedef enum
294: {
295:     IDI_BANK_INFO_DEFINITION( BANK_EXTRACT_ENUM )
296: } IDI_BANK_ENUM;
297:
298: typedef enum
299: {
300:     IDO_BANK_INFO_DEFINITION( BANK_EXTRACT_ENUM )
301: } IDO_BANK_ENUM;
302:
303: typedef enum
304: {
305:     BANK_INFO_DEFINITION( BANK_EXTRACT_ENUM )
306: } BANK_ENUM;
307:
308:
309:
310:
311: #define BANK_NULL_DEFINITION           { BANK_UNDEFINED,    NULL      }
312:
313:
314:
315: struct idi_bank_info
316: {
317:     int      symbol;
318:     const char * name;
319: };
320:
321: struct ido_bank_info
322: {
323:     IDI_BANK_ENUM symbol;
324:     const char * name;
325: };
326:
327: struct bank_info
328: {
329:     BANK_ENUM   symbol;
330:     const char * name;
331: };
332:
333:
334:
335: typedef enum
336: {
337:     REG_DIR_NONE    = 0x00,
338:     REG_DIR_READ    = 0x01,
339:     REG_DIR_WRITE    = 0x02,
340:     REG_DIR_READ_WRITE = 0x03
341: } REG_DIR_ENUM;
342:
343:
344:
345:
352: #define IDI_REGISTER_SET_DEFINITION(_) \
353: \
354: \
355: \
```

```

356: _-( IDI_DI_GROUP0,      0,  0,  1,  1,  REG_DIR_READ, "dig0",  IDI_BANK_NONE ) \
357: _-( IDI_DI_GROUP1,      1,  1,  1,  1,  REG_DIR_READ, "dig1",  IDI_BANK_NONE ) \
358: _-( IDI_DI_GROUP2,      2,  2,  1,  1,  REG_DIR_READ, "dig2",  IDI_BANK_NONE ) \
359: _-( IDI_DI_GROUP3,      3,  3,  1,  1,  REG_DIR_READ, "dig3",  IDI_BANK_NONE ) \
360: _-( IDI_DI_GROUP4,      4,  4,  1,  1,  REG_DIR_READ, "dig4",  IDI_BANK_NONE ) \
361: _-( IDI_DI_GROUP5,      5,  5,  1,  1,  REG_DIR_READ, "dig5",  IDI_BANK_NONE ) \
362: \
363: _-( IDI_INTR_BY_GROUP,   6,  6,  1,  1,  REG_DIR_READ, "isbg",  IDI_BANK_NONE ) \
364: \
365: _-( IDI_PEND_GROUP0,     7,  8,  1,  1,  REG_DIR_READ, "p0",    IDI_BANK_3 ) \
366: _-( IDI_PEND_GROUP1,     8,  9,  1,  1,  REG_DIR_READ, "p1",    IDI_BANK_3 ) \
367: _-( IDI_PEND_GROUP2,     9, 10,  1,  1,  REG_DIR_READ, "p2",    IDI_BANK_3 ) \
368: _-( IDI_PEND_GROUP3,    10, 11,  1,  1,  REG_DIR_READ, "p3",    IDI_BANK_3 ) \
369: _-( IDI_PEND_GROUP4,    11, 12,  1,  1,  REG_DIR_READ, "p4",    IDI_BANK_3 ) \
370: _-( IDI_PEND_GROUP5,    12, 13,  1,  1,  REG_DIR_READ, "p5",    IDI_BANK_3 ) \
371: _-( IDI_CLEAR_GROUP0,    13,  8,  1,  1,  REG_DIR_WRITE, "c0",   IDI_BANK_3 ) \
372: _-( IDI_CLEAR_GROUP1,    14,  9,  1,  1,  REG_DIR_WRITE, "c1",   IDI_BANK_3 ) \
373: _-( IDI_CLEAR_GROUP2,    15, 10,  1,  1,  REG_DIR_WRITE, "c2",   IDI_BANK_3 ) \
374: _-( IDI_CLEAR_GROUP3,    16, 11,  1,  1,  REG_DIR_WRITE, "c3",   IDI_BANK_3 ) \
375: _-( IDI_CLEAR_GROUP4,    17, 12,  1,  1,  REG_DIR_WRITE, "c4",   IDI_BANK_3 ) \
376: _-( IDI_CLEAR_GROUP5,    18, 13,  1,  1,  REG_DIR_WRITE, "c5",   IDI_BANK_3 ) \
377: \
378: _-( IDI_BANK,      19,  7,  1,  1,  REG_DIR_READ_WRITE, "bank",  IDI_BANK_NONE ) \
379: _-( IDI_ID_LSB,     20, 14,  1,  1,  REG_DIR_READ, "idlsb",  IDI_BANK_NONE ) \
380: _-( IDI_ID_MSB,     21, 15,  1,  1,  REG_DIR_READ, "idmsb",  IDI_BANK_NONE ) \
381: _-( IDI_ZERO0,     22,  8,  1,  1,  REG_DIR_READ, "zbo0",   IDI_BANK_0 ) \
382: _-( IDI_ZERO1,     23,  9,  1,  1,  REG_DIR_READ, "zbl1",   IDI_BANK_0 ) \
383: _-( IDI_ZERO2,     24, 10,  1,  1,  REG_DIR_READ, "zb2",   IDI_BANK_0 ) \
384: _-( IDI_ZERO3,     25, 11,  1,  1,  REG_DIR_READ, "zb3",   IDI_BANK_0 ) \
385: _-( IDI_ZERO4,     26, 12,  1,  1,  REG_DIR_READ, "zb4",   IDI_BANK_0 ) \
386: _-( IDI_ZERO5,     27, 13,  1,  1,  REG_DIR_READ, "zb5",   IDI_BANK_0 ) \
387: \
388: _-( IDI_FPGA_DATA,   28,  8,  1,  1,  REG_DIR_READ, "fd",    IDI_BANK_0 ) \
389: _-( IDI_FPGA_INDEX,  29,  9,  1,  1,  REG_DIR_READ_WRITE, "fi",    IDI_BANK_0 ) \
390: \
391: _-( IDI_EDGE_GROUP0,   30,  8,  1,  1,  REG_DIR_READ_WRITE, "ep0",  IDI_BANK_1 ) \
392: _-( IDI_EDGE_GROUP1,   31,  9,  1,  1,  REG_DIR_READ_WRITE, "ep1",  IDI_BANK_1 ) \
393: _-( IDI_EDGE_GROUP2,   32, 10,  1,  1,  REG_DIR_READ_WRITE, "ep2",  IDI_BANK_1 ) \
394: _-( IDI_EDGE_GROUP3,   33, 11,  1,  1,  REG_DIR_READ_WRITE, "ep3",  IDI_BANK_1 ) \
395: _-( IDI_EDGE_GROUP4,   34, 12,  1,  1,  REG_DIR_READ_WRITE, "ep4",  IDI_BANK_1 ) \
396: _-( IDI_EDGE_GROUP5,   35, 13,  1,  1,  REG_DIR_READ_WRITE, "ep5",  IDI_BANK_1 ) \
397: _-( IDI_INTR_BIT_GROUP0, 36,  8,  1,  1,  REG_DIR_READ_WRITE, "ibe0",  IDI_BANK_2 ) \
398: _-( IDI_INTR_BIT_GROUP1, 37,  9,  1,  1,  REG_DIR_READ_WRITE, "ibel",  IDI_BANK_2 ) \
399: _-( IDI_INTR_BIT_GROUP2, 38, 10,  1,  1,  REG_DIR_READ_WRITE, "ibe2",  IDI_BANK_2 ) \
) \
400: _-( IDI_INTR_BIT_GROUP3, 39, 11,  1,  1,  REG_DIR_READ_WRITE, "ibe3",  IDI_BANK_2 ) \
) \
401: _-( IDI_INTR_BIT_GROUP4, 40, 12,  1,  1,  REG_DIR_READ_WRITE, "ibe4",  IDI_BANK_2 ) \
) \
402: _-( IDI_INTR_BIT_GROUP5, 41, 13,  1,  1,  REG_DIR_READ_WRITE, "ibe5",  IDI_BANK_2 ) \
) \
403: \
404: _-( IDI_SPI_ID_LSB,    42,  8,  1,  1,  REG_DIR_READ, "sidlsb", IDI_BANK_6 ) \
405: _-( IDI_SPI_ID_MSB,    43,  9,  1,  1,  REG_DIR_READ, "sidmsb", IDI_BANK_6 ) \
406: _-( IDI_SPI_CONFIG,   44, 10,  1,  1,  REG_DIR_READ_WRITE, "scfg",  IDI_BANK_6 ) \
407: _-( IDI_SPI_ECD,      45, 11,  1,  1,  REG_DIR_READ_WRITE, "secd",  IDI_BANK_6 ) \
408: _-( IDI_SPI_HCI_LSB,   46, 12,  1,  1,  REG_DIR_READ_WRITE, "shclsb", IDI_BANK_6 ) \
409: _-( IDI_SPI_HCI_MSB,   47, 13,  1,  1,  REG_DIR_READ_WRITE, "shcmsb", IDI_BANK_6 ) \
410: _-( IDI_SPI_DATA,     48,  8,  1,  1,  REG_DIR_READ_WRITE, "sdata",  IDI_BANK_7 ) \
411: _-( IDI_SPI_TX_STATUS, 49,  9,  1,  1,  REG_DIR_READ, "stxf",   IDI_BANK_7 ) \
412: _-( IDI_SPI_RX_STATUS, 50, 10,  1,  1,  REG_DIR_READ, "srxf",   IDI_BANK_7 ) \
413: _-( IDI_SPI_COMMIT,   51, 11,  1,  1,  REG_DIR_READ_WRITE, "scmt",  IDI_BANK_7 ) \
414: _-( IDI_UNDEFINED,    52,  0,  0,  0,  REG_DIR_NONE, "",      IDI_BANK_UNDEFINED ) \
415:

```

```

416:
417: #define IDO_REGISTER_SET_DEFINITION(_)
418: \
419: \
420: \
421: _(_ IDO_DO_GROUP0,      0, 0, 1, 1, REG_DIR_READ_WRITE, "dog0", IDO_BANK_NONE ) \
422: _(_ IDO_DO_GROUP1,      1, 1, 1, 1, REG_DIR_READ_WRITE, "dog1", IDO_BANK_NONE ) \
423: _(_ IDO_DO_GROUP2,      2, 2, 1, 1, REG_DIR_READ_WRITE, "dog2", IDO_BANK_NONE ) \
424: _(_ IDO_DO_GROUP3,      3, 3, 1, 1, REG_DIR_READ_WRITE, "dog3", IDO_BANK_NONE ) \
425: _(_ IDO_DO_GROUP4,      4, 4, 1, 1, REG_DIR_READ_WRITE, "dog4", IDO_BANK_NONE ) \
426: _(_ IDO_DO_GROUP5,      5, 5, 1, 1, REG_DIR_READ_WRITE, "dog5", IDO_BANK_NONE ) \
427: \
428: _(_ IDO_BANK,          6, 7, 1, 1, REG_DIR_READ_WRITE, "bank", IDO_BANK_NONE ) \
429: \
430: _(_ IDO_FPGA_DATA,     7, 8, 1, 1, REG_DIR_READ, "fd", IDI_BANK_0 ) \
431: _(_ IDO_FPGA_INDEX,    8, 9, 1, 1, REG_DIR_READ_WRITE, "fi", IDI_BANK_0 ) \
432: \
433: _(_ IDO_ID_LSB,        9, 14, 1, 1, REG_DIR_READ, "idlsb", IDO_BANK_NONE ) \
434: _(_ IDO_ID_MSB,        10, 15, 1, 1, REG_DIR_READ, "idmsb", IDO_BANK_NONE ) \
435: \
436: _(_ IDO_SPI_ID_LSB,   11, 8, 1, 1, REG_DIR_READ, "sidlsb", IDO_BANK_6 ) \
437: _(_ IDO_SPI_ID_MSB,   12, 9, 1, 1, REG_DIR_READ, "sidmsb", IDO_BANK_6 ) \
438: _(_ IDO_SPI_CONFIG,   13, 10, 1, 1, REG_DIR_READ_WRITE, "scfg", IDO_BANK_6 ) \
439: _(_ IDO_SPI_ECD,      14, 11, 1, 1, REG_DIR_READ_WRITE, "secd", IDO_BANK_6 ) \
440: _(_ IDO_SPI_HCI_LSB,  15, 12, 1, 1, REG_DIR_READ_WRITE, "shclsb", IDO_BANK_6 ) \
441: _(_ IDO_SPI_HCI_MSB,  16, 13, 1, 1, REG_DIR_READ_WRITE, "shcmsb", IDO_BANK_6 ) \
442: _(_ IDO_SPI_DATA,     17, 8, 1, 1, REG_DIR_READ_WRITE, "sdata", IDO_BANK_7 ) \
443: _(_ IDO_SPI_TX_STATUS, 18, 9, 1, 1, REG_DIR_READ, "stxf", IDO_BANK_7 ) \
444: _(_ IDO_SPI_RX_STATUS, 19, 10, 1, 1, REG_DIR_READ, "srxf", IDO_BANK_7 ) \
445: _(_ IDO_SPI_COMMIT,   20, 11, 1, 1, REG_DIR_READ_WRITE, "scmt", IDO_BANK_7 ) \
446: _(_ IDO_UNDEFINED,    21, 0, 0, 0, REG_DIR_NONE, "", IDO_BANK_UNDEFINED ) \
447:
448:
449:
450:
451: #define IAI_REGISTER_SET_DEFINITION(_)
452: \
453: \
454: \
455: _(_ IAI_AIN0,          0, 0x00, 2, 1, REG_DIR_READ, "ail", IAI_BANK_NONE ) \
456: _(_ IAI_AIN1,          1, 0x02, 2, 1, REG_DIR_READ, "ai2", IAI_BANK_NONE ) \
457: _(_ IAI_AIN2,          2, 0x04, 2, 1, REG_DIR_READ, "ai3", IAI_BANK_NONE ) \
458: _(_ IAI_AIN3,          3, 0x06, 2, 1, REG_DIR_READ, "ai4", IAI_BANK_NONE ) \
459: _(_ IAI_AIN4,          4, 0x20, 2, 1, REG_DIR_READ, "ai5", IAI_BANK_NONE ) \
460: _(_ IAI_AIN5,          5, 0x22, 2, 1, REG_DIR_READ, "ai6", IAI_BANK_NONE ) \
461: _(_ IAI_AIN6,          6, 0x24, 2, 1, REG_DIR_READ, "ai7", IAI_BANK_NONE ) \
462: _(_ IAI_AIN7,          7, 0x26, 2, 1, REG_DIR_READ, "ai8", IAI_BANK_NONE ) \
463: _(_ IAI_AIN8,          8, 0x40, 2, 1, REG_DIR_READ, "ai9", IAI_BANK_NONE ) \
464: _(_ IAI_AIN9,          9, 0x42, 2, 1, REG_DIR_READ, "ai10", IAI_BANK_NONE ) \
465: _(_ IAI_AIN10,         10, 0x44, 2, 1, REG_DIR_READ, "ai11", IAI_BANK_NONE ) \
466: _(_ IAI_AIN11,         11, 0x46, 2, 1, REG_DIR_READ, "ai12", IAI_BANK_NONE ) \
467: _(_ IAI_AIN12,         12, 0x60, 2, 1, REG_DIR_READ, "ai13", IAI_BANK_NONE ) \
468: _(_ IAI_AIN13,         13, 0x62, 2, 1, REG_DIR_READ, "ai14", IAI_BANK_NONE ) \
469: _(_ IAI_AIN14,         14, 0x64, 2, 1, REG_DIR_READ, "ai15", IAI_BANK_NONE ) \
470: _(_ IAI_AIN15,         15, 0x66, 2, 1, REG_DIR_READ, "ai16", IAI_BANK_NONE ) \
471: \
472: _(_ IAI_FPGA_DATA,     7, 8, 1, 1, REG_DIR_READ, "fd", IDI_BANK_0 ) \
473: _(_ IAI_FPGA_INDEX,    8, 9, 1, 1, REG_DIR_READ_WRITE, "fi", IDI_BANK_0 ) \
474: \
475: _(_ IAI_ID_LSB,        17, 0x6E, 1, 1, REG_DIR_READ, "idlsb", IAI_BANK_NONE ) \
476: _(_ IAI_ID_MSB,        18, 0x6F, 1, 1, REG_DIR_READ, "idmsb", IAI_BANK_NONE ) \
477: \
478: _(_ IAI_SPI_ID_LSB,   19, 0x70, 1, 1, REG_DIR_READ, "sidlsb", IAI_BANK_NONE ) \
479: _(_ IAI_SPI_ID_MSB,   20, 0x71, 1, 1, REG_DIR_READ, "sidmsb", IAI_BANK_NONE ) \

```

```

480: _-( IAI_SPI_CONFIG,    21, 0x72, 1, 1, REG_DIR_READ_WRITE, "scfg", IAI_BANK_NONE ) \
481: _-( IAI_SPI_ECD,      22, 0x73, 1, 1, REG_DIR_READ_WRITE, "secd", IAI_BANK_NONE ) \
482: _-( IAI_SPI_HCI_LSB,   23, 0x74, 1, 1, REG_DIR_READ_WRITE, "shclsb", IAI_BANK_NONE
) \
483: _-( IAI_SPI_HCI_MSB,   24, 0x75, 1, 1, REG_DIR_READ_WRITE, "shcmsb", IAI_BANK_NONE
) \
484: _-( IAI_SPI_DATA,     25, 0x76, 1, 1, REG_DIR_READ_WRITE, "sdata", IAI_BANK_NONE ) \
485: _-( IAI_SPI_TX_STATUS, 26, 0x77, 1, 1, REG_DIR_READ, "stxf", IAI_BANK_NONE ) \
486: _-( IAI_SPI_RX_STATUS, 27, 0x78, 1, 1, REG_DIR_READ, "srxf", IAI_BANK_NONE ) \
487: _-( IAI_SPI_COMMIT,   28, 0x79, 1, 1, REG_DIR_READ_WRITE, "scmt", IAI_BANK_NONE ) \
488: _-( IAI_UNDEFINED,    29, 0, 0, 0, REG_DIR_NONE, "", IAI_BANK_UNDEFINED )
489:
490: #if(0)
491: #define SPI_REGISTER_SET_DEFINITION(_)
492: \
493: \
494: \
495: _-( SPI_ID_LSB,        7, 8, 1, 1, REG_DIR_READ, "sidlsb", BANK_6 ) \
496: _-( SPI_ID_MSB,        8, 9, 1, 1, REG_DIR_READ, "sidmsb", BANK_6 ) \
497: _-( SPI_CONFIG,        9, 10, 1, 1, REG_DIR_READ_WRITE, "scfg", BANK_6 ) \
498: _-( SPI_ECD,          10, 11, 1, 1, REG_DIR_READ_WRITE, "secd", BANK_6 ) \
499: _-( SPI_HCI_LSB,       11, 12, 1, 1, REG_DIR_READ_WRITE, "shclsb", BANK_6 ) \
500: _-( SPI_HCI_MSB,       12, 13, 1, 1, REG_DIR_READ_WRITE, "shcmsb", BANK_6 ) \
501: _-( SPI_DATA,          13, 8, 1, 1, REG_DIR_READ_WRITE, "sdata", BANK_7 ) \
502: _-( SPI_TX_STATUS,     14, 9, 1, 1, REG_DIR_READ, "stxf", BANK_7 ) \
503: _-( SPI_RX_STATUS,     15, 10, 1, 1, REG_DIR_READ, "srxf", BANK_7 ) \
504: _-( SPI_COMMIT,        16, 11, 1, 1, REG_DIR_READ_WRITE, "scmt", BANK_7 ) \
505: _-( SPI_UNDEFINED,     17, 0, 0, 0, REG_DIR_NONE, "", BANK_UNDEFINED )
506: #endif
507:
508:
509: #define REG_LOCATION_LOGICAL_GET(location) ( ( (location) >> 8 ) & 0xFF )
510: #define REG_LOCATION_CHANNEL_GET(location) ( (location) & 0xFF )
511:
512:
520: #define REG_LOCATION_SET(index,channel) ( ( ((index) & 0xFF) << 8 ) | ((channel)
& 0xFF) )
521:
522:
526: #define
REG_EXTRACT_ENUM(symbol,index,offset,register_bytes,aperture_bytes,read_write,name,bank)
symbol = REG_LOCATION_SET(index,0),
527:
528:
532: #define
REG_EXTRACT_DEFINITION(symbol,index,offset,register_bytes,aperture_bytes,read_write,name,bank)
{ REG_LOCATION_SET(index,0), read_write, bank, offset, #symbol, name },
533:
534:
542:
543: typedef enum
544: {
545: IDI_REGISTER_SET_DEFINITION( REG_EXTRACT_ENUM )
546: } IDI_REG_ENUM;
547:
548:
549: typedef enum
550: {
551: IDO_REGISTER_SET_DEFINITION( REG_EXTRACT_ENUM )
552: } IDO_REG_ENUM;
553:
554:
555: struct idi_reg_definition
556: {

```

```
557: IDI_REG_ENUM symbol;
558: REG_DIR_ENUM direction;
559: IDI_BANK_ENUM bank;
560: uint16_t physical_offset;
561: char * symbol_name;
562: char * acronym;
563: };
564:
565:
566: struct ido_reg_definition
567: {
568: IDO_REG_ENUM symbol;
569: REG_DIR_ENUM direction;
570: IDO_BANK_ENUM bank;
571: uint16_t physical_offset;
572: char * symbol_name;
573: char * acronym;
574: };
575:
576:
577:
578: struct reg_definition
579: {
580: int symbol;
581: REG_DIR_ENUM direction;
582: int bank;
583: uint16_t physical_offset;
584: char * symbol_name;
585: char * acronym;
586: };
587:
588:
589: typedef enum
590: {
591: CSB_SOFTWARE = 0,
592: CSB_BUFFER = 1,
593: CSB_uint8_t = 2,
594: CSB_uint16_t = 3
595: } SPI_CSBOARD_ENUM;
596:
597:
598:
599: enum { SPI_FIFO_SIZE = 8 };
600:
601: enum { SPI_FIFO_SIZE_BIT_WIDTH = 15 };
602:
603:
604:
605: struct spi_cfg
606: {
607: BOOL sdio_wrap;
608: BOOL sdo_polarity;
609: BOOL sdi_polarity;
610: BOOL sclk_phase;
611: BOOL sclk_polarity;
612: BOOL chip_select_route_ch1;
613: SPI_CSBOARD_ENUM chip_select_behavior;
614: uint8_t end_cycle_delay;
615: uint16_t half_clock_interval;
616:
617: double clock_hz;
618: double end_delay_ns;
619: };
620:
```

```
621: struct spi_status
622: {
623:   BOOL tx_status;
624:   BOOL full;
625:   BOOL empty;
626:   int fifo_size;
627:   int fifo_count;
628: };
629:
630:
631:
632: #define IDI_DIN_GROUP_SIZE 8
633: #define IDI_DIN_SHIFT_RIGHT 3
634: #define IDI_DIN_GROUP_QTY 6
635: #define IDI_DIN_QTY ( IDI_DIN_GROUP_SIZE * IDI_DIN_GROUP_QTY )
636:
637:
638: #define IDO_DO_GROUP_SIZE 8
639: #define IDO_DO_SHIFT_RIGHT 3
640: #define IDO_DO_GROUP_QTY 6
641: #define IDO_DO_QTY ( IDO_DO_GROUP_SIZE * IDO_DO_GROUP_QTY )
642:
643:
644: struct din_cfg
645: {
646:   struct
647:   {
648:     BOOL falling_edge;
649:     BOOL interrupt_enable;
650:   } chan[IDI_DIN_QTY];
651: };
652:
653: struct dout_cfg
654: {
655:   struct
656:   {
657:     BOOL value;
658:   } chan[IDO_DO_QTY];
659: };
660:
661:
662:
663:
664:
665: #define FRAM_BLOCK_SIZE 256
666: #define SPI_BLOCK_SIZE 256
667: #define REGS_INIT_QTY 16
668: #define MESSAGE_SIZE 256
669:
670:
671:
672:
673: struct spi_dataset
674: {
675:   uint16_t id;
676:   struct spi_cfg cfg;
677:
678:
679:   uint8_t fram_block[FRAM_BLOCK_SIZE];
680:
681:
682:   uint8_t tx_buffer[SPI_BLOCK_SIZE];
683:   uint8_t rx_buffer[SPI_BLOCK_SIZE];
684: };
685:
686:
687:
688:
```

```

689:
690: #define IO_TRACE_DEFINITION(_) \
691: \
692: _(_( IO_TRACE_NULL, 0, "NULL" ) \
693: _(_( IO_TRACE_READ, 1, "IO_Read_U8" ) \
694: _(_( IO_TRACE_WRITE, 2, "IO_Write_U8" ) \
695: _(_( IO_TRACE_SPI, 3, "SPI" ) \
696: _(_( IO_TRACE_FRAM, 4, "FRAM" ) \
697: _(_( IO_TRACE_AS, 5, "Analog_Stick" ) \
698: _(_( IO_TRACE_OTHER, 6, "Other" ) \
699:
700: #define IO_TRACE_EXTRACT_ENUM(symbol,index,name) symbol = index,
701: #define IO_TRACE_EXTRACT_NAME(symbol,index,name) name,
702:
703:
704:
705:
706:
707:
708:
709:
710:
711:
712:
713:
714:
715:
716:
717: #define IO_TRACE_START_DEFINITION(_) \
718: \
719: _(_( IO_TRACE_START_DISABLE, 0, "disable", ds->trace.active = false; ) \
720: _(_( IO_TRACE_START_ON, 1, "on", ds->trace.active = true; ) \
721: _(_( IO_TRACE_START_ERROR, 2, "error", if ( SUCCESS != error_code ) \
ds->trace.active = true; ) \
722: _(_( IO_TRACE_START_SPI, 3, "spi", if ( IO_TRACE_SPI == action ) \
ds->trace.active = true; ) \
723: _(_( IO_TRACE_START_FRAM, 4, "fram", if ( IO_TRACE_FRAM == action ) \
ds->trace.active = true; ) \
724: _(_( IO_TRACE_START_AS, 5, "as", if ( IO_TRACE_AS == action ) \
ds->trace.active = true; ) \
725:
726:
727: #define IO_TRACE_STOP_DEFINITION(_) \
728: \
729: _(_( IO_TRACE_STOP_NONE, 0, "none" ) \
730: _(_( IO_TRACE_STOP_ERROR, 1, "error", if ( SUCCESS != error_code ) ds->trace.active = \
false; ) \
731: _(_( IO_TRACE_STOP_CTRLC, 2, "ctrlc", \
732:
733:
734:
735: #define IO_TRACE_START_STOP_EXTRACT_ENUM(symbol,index,name,action) symbol = index,
736: #define IO_TRACE_START_STOP_EXTRACT_NAME(symbol,index,name,action) name,
737:
738: #define IO_TRACE_START_STOP_EXTRACT_NAME_HELP_LIST(symbol,index,name,action) name " "
739:
740: #define IO_TRACE_START_STOP_EXTRACT_SWITCH_STATEMENT(symbol,index,name,action) \
741: case symbol: \
742: { action } \
743: break;
744:
745:
746: enum IO_TRACE_ENUM
747: {
748: IO_TRACE_DEFINITION( IO_TRACE_EXTRACT_ENUM )
749: };

```

```
750:
751: enum IO_TRACE_START_ENUM
752: {
753:     IO_TRACE_START_DEFINITION( IO_TRACE_START_STOP_EXTRACT_ENUM )
754: };
755:
756: enum IO_TRACE_STOP_ENUM
757: {
758:     IO_TRACE_STOP_DEFINITION( IO_TRACE_START_STOP_EXTRACT_ENUM )
759: };
760:
761: #define IO_TRACE_SIZE      1024
762: #define IO_TRACE_FILE_NAME_SIZE 32
763: #define IO_TRACE_DUMP_FILE_NAME "iotrace.csv"
764:
765: #define IO_TRACE_USE_SPI_LOGGING    1
766: #define IO_TRACE_USE_FRAM_LOGGING   1
767: #define IO_TRACE_USE_AS_LOGGING    1
768:
769: struct io_trace
770: {
771:     int      error_code;
772:     enum IO_TRACE_ENUM  action;
773:     int      board_id;
774:     int      location;
775:     size_t   line;
776:     size_t   posting;
777:     uint8_t  value;
778:     const char *  function_name;
779: };
780:
781: struct io_trace_info
782: {
783:     size_t      begin;
784:     size_t      end;
785:     size_t      index;
786:     size_t      count;
787:     size_t      posting;
788:     BOOL        wrap;
789:     enum IO_TRACE_START_ENUM start;
790:     enum IO_TRACE_STOP_ENUM  stop;
791:     BOOL        active;
792:     char        filename[IO_TRACE_FILE_NAME_SIZE];
793:     struct io_trace *  buf;
794: };
795:
796:
797: typedef enum
798: {
799:     MODE_JUMPERS_0  = 0,
800:     MODE_JUMPERS_1  = 1,
801:     MODE_JUMPERS_2  = 2,
802:     MODE_JUMPERS_3  = 3
803: } MODE_JUMPERS_ENUM;
804:
805:
806:
810: #define AS_AM2315_REGISTER_QTY  9
811:
812: struct as_am2315_registers
813: {
814:     uint8_t  reg[AS_AM2315_REGISTER_QTY];
815: };
816:
```

```
817:  
818: struct idi_dataset  
819: {  
820:  
821:   int      board_id;  
822:   BOOL      set__suppress_io_activity;  
823:  
824:   BOOL      quit_application;  
825:   uint16_t   base_address;  
826:   IDI_BANK_ENUM bank_previous;  
827:  
828:   BOOL      io_simulate;  
829:   BOOL      io_report;  
830:   BOOL      spi_is_present;  
831:  
832:   BOOL      fram_cs_default;  
833:   BOOL      fram_spi_cs_route_backup;  
834:   BOOL      as_cs_default;  
835:   BOOL      as_spi_cs_route_backup;  
836:   MODE_JUMPERS_ENUM mode_jumpers;  
837:   const char *    svn_revision_string;  
838:   struct spi_dataset   spi;  
839:   struct io_trace_info   trace;  
840:   struct as_ads1259_registers   ads1259;  
841:  
842:  
843:   char        message[MESSAGE_SIZE];  
844:  
845:  
846:  
847:   unsigned int   irq_number;  
848:  
849:  
850:   BOOL      irq.Please_install_handler_request;  
851:   BOOL      irq.handler_active;  
852:   size_t    irq.quantity;  
853:   size_t    volatile  irq.count;  
854:   size_t    irq.count_previous;  
855:  
856:  
857:   uint8_t   volatile  isr_pending_list[IDI_DIN_GROUP_QTY];  
858:  
859:  
860:  
861:  
862:  
863:  
864:  
865:   struct din_cfg   din_cfg;  
866:  
867:   struct  
868:   {  
869:     BOOL      used;  
870:     int       address;  
871:     IDI_REG_ENUM location;  
872:     uint8_t   value;  
873:   } reg_init[REGS_INIT_QTY];  
874: };  
875:  
876:  
877:   struct ido_dataset  
878: {  
879:  
880:   int      board_id;  
881:   BOOL      set__suppress_io_activity;  
882:  
883:   BOOL      quit_application;  
884:   uint16_t   base_address;  
885:   IDI_BANK_ENUM bank_previous;  
886:
```

```
887: BOOL      io_simulate;
888: BOOL      io_report;
889: BOOL      spi_is_present;
890:
891: BOOL      fram_cs_default;
892: BOOL      fram_spi_cs_route_backup;
893: BOOL      as_cs_default;
894: BOOL      as_spi_cs_route_backup;
895: MODE_JUMPERS_ENUM mode_jumpers;
896: const char *    svn_revision_string;
897: struct spi_dataset   spi;
898: struct io_trace_info  trace;
899: struct as_ads1259_registers ads1259;
900:
901: char        message[MESSAGE_SIZE];
903:
904:
905: unsigned int   irq_number;
906:
909: BOOL      irq.Please_install_handler_request;
910: BOOL      irq.handler_active;
911: size_t     irq.quantity;
912: size_t volatile irq_count;
913: size_t     irq_count_previous;
914:
919: uint8_t volatile isr_pending_list[IDI_DIN_GROUP_QTY];
920:
921:
922:
923: struct dout_cfg  dout_cfg;
924:
925: struct
926: {
927:     BOOL    used;
928:     int     address;
929:     IDO_REG_ENUM location;
930:     uint8_t  value;
931: } reg_init[REGS_INIT_QTY];
932:
933: };
934:
935:
940: struct board_dataset
941: {
942:
943:     int      board_id;
944:     BOOL    set__suppress_io_activity;
945:
946:     BOOL    quit_application;
947:     uint16_t base_address;
948:     int     bank_previous;
949:     BOOL    io_simulate;
950:     BOOL    io_report;
951:     BOOL    spi_is_present;
952:
953:     BOOL    fram_cs_default;
954:     BOOL    fram_spi_cs_route_backup;
955:     BOOL    as_cs_default;
956:     BOOL    as_spi_cs_route_backup;
957:     MODE_JUMPERS_ENUM mode_jumpers;
958:     const char *    svn_revision_string;
959:     struct spi_dataset   spi;
960:     struct io_trace_info  trace;
```

```
961: struct as_ads1259_registers ads1259;
962: };
963:
964:
969:
970:
971: static const char svn_revision_string[] = { SVN_REV };
972:
973:
974: const struct ec_human_readable ec_human_readable[] =
975: {
976:     ERROR_CODES( EC_EXTRACT_HUMAN_READABLE )
977:     EC_HUMAN_READABLE_TERMINATE
978: };
979:
980:
981:
982: static const struct idi_bank_info idi_bank_info[] =
983: {
984:     IDI_BANK_INFO_DEFINITION( BANK_EXTRACT_DEFINITION )
985:     BANK_NULL_DEFINITION
986: };
987:
988: static const struct ido_bank_info ido_bank_info[] =
989: {
990:     IDO_BANK_INFO_DEFINITION( BANK_EXTRACT_DEFINITION )
991:     BANK_NULL_DEFINITION
992: };
993:
994:
995:
996: static const struct idi_reg_definition idi_reg_definition[] =
997: {
998:     IDI_REGISTER_SET_DEFINITION( REG_EXTRACT_DEFINITION )
999: };
1000:
1001: static const struct ido_reg_definition ido_reg_definition[] =
1002: {
1003:     IDO_REGISTER_SET_DEFINITION( REG_EXTRACT_DEFINITION )
1004: };
1005:
1006:
1009: static struct idi_dataset idi_ds;
1010:
1011:
1012: static struct ido_dataset ido_ds;
1013:
1014:
1015: static struct io_trace global_io_trace_buf[IO_TRACE_SIZE];
1016: static BOOL global_io_trace_enable;
1017: static const char * global_io_trace_start_name[] = { IO_TRACE_START_DEFINITION(
IO_TRACE_START_STOP_EXTRACT_NAME )
1018:             NULL
1019:             };
1020: static const char * global_io_trace_stop_name[] = { IO_TRACE_STOP_DEFINITION(
IO_TRACE_START_STOP_EXTRACT_NAME )
1021:             NULL
1022:             };
1023: static const char * global_io_trace_name[] = { IO_TRACE_DEFINITION(
IO_TRACE_EXTRACT_NAME )
1024:             NULL
1025:             };
1026:
1027:
```

```
1032:
1033: #define IDI_BOARD_INIT_FILE_NAME      "idi_init.bin"
1034: #define IDO_BOARD_INIT_FILE_NAME      "ido_init.bin"
1035:
1036: struct board_definition
1037: {
1038:     const void * bank_info;
1039:     const void * definition;
1040:     const void * dataset;
1041:     const int    bank_register_symbol;
1042:     const int    fpga_data_symbol;
1043:     const int    fpga_index_symbol;
1044: };
1045:
1046: static struct board_definition board_definition[] =
1047: {
1048:     { (const void *) idi_bank_info, (const void *) idi_reg_definition, (const void *)
1049: &idi_ds, IDI_BANK, IDI_FPGA_DATA, IDI_FPGA_INDEX },
1050:     { (const void *) ido_bank_info, (const void *) ido_reg_definition, (const void *)
1051: &ido_ds, IDO_BANK, IDO_FPGA_DATA, IDO_FPGA_INDEX },
1052:     { NULL,        NULL,        NULL,        IDO_BANK_NONE, 0, 0 }
1053: };
1054: static int  global_board_id          = ID_IDI48;
1055: static BOOL global_loop_command     = false;
1056: static int  global_loop_count       = 0;
1057: static int  global_loop_count_counter = 0;
1058: static int  global_loop_delay_ms    = 0;
1059: static BOOL global_loop_space      = false;
1060: static BOOL global_irq_please_install_handler_request = false;
1061:
1062:
1063:
1064:
1065:
1066:
1067:
1068:
1069:
1070:
1071:
1072:
1073:
1074:
1082: static const char * IDI_Bank_Symbol_To_Name( struct idi_bank_info * info, IDI_BANK_ENUM
bank )
1083: {
1084:     int index;
1085:
1086:     switch( bank )
1087:     {
1088:         case IDI_BANK_0:   index = 0; break;
1089:         case IDI_BANK_1:   index = 1; break;
1090:         case IDI_BANK_2:   index = 2; break;
1091:         case IDI_BANK_3:   index = 3; break;
1092:         case IDI_BANK_4:   index = 4; break;
1093:         case IDI_BANK_5:   index = 5; break;
1094:         case IDI_BANK_6:   index = 6; break;
1095:         case IDI_BANK_7:   index = 7; break;
1096:         case IDI_BANK_NONE: index = 8; break;
1097:         case IDI_BANK_UNDEFINED: index = 9; break;
1098:     }
1099:     return info[index].name;
1100: }
1101:
1102: static const char * IDO_Bank_Symbol_To_Name( struct ido_bank_info * info, IDO_BANK_ENUM
```

```
bank )
1103: {
1104:     int index;
1105:
1106:     switch( bank )
1107:     {
1108:         case IDO_BANK_0:    index = 0; break;
1109:         case IDO_BANK_1:    index = 1; break;
1110:         case IDO_BANK_2:    index = 2; break;
1111:         case IDO_BANK_3:    index = 3; break;
1112:         case IDO_BANK_4:    index = 4; break;
1113:         case IDO_BANK_5:    index = 5; break;
1114:         case IDO_BANK_6:    index = 6; break;
1115:         case IDO_BANK_7:    index = 7; break;
1116:         case IDO_BANK_NONE: index = 8; break;
1117:         case IDO_BANK_UNDEFINED: index = 9; break;
1118:     }
1119:     return info[index].name;
1120: }
1121:
1122:
1123: static const char * Bank_Symbol_To_Name( int board_id, int bank )
1124: {
1125:     switch ( board_id )
1126:     {
1127:         case ID_IDI48: return IDI_Bank_Symbol_To_Name( ((struct idi_bank_info *)
board_definition[board_id].bank_info), (IDI_BANK_ENUM) bank );
1128:         case ID_IDO48: return IDO_Bank_Symbol_To_Name( ((struct ido_bank_info *)
board_definition[board_id].bank_info), (IDO_BANK_ENUM) bank );
1129:     }
1130:     return NULL;
1131: }
1132:
1133:
1134:
1135:
1142:
1143: static int IDI_Bank_Name_To_Symbol( int board_id, const char * name, IDI_BANK_ENUM *
bank )
1144: {
1145:     int index;
1146:     const struct bank_info * info = board_definition[board_id].bank_info;
1147:     index = 0;
1148:     while ( NULL != info[index].name )
1149:     {
1150:         if ( 0 == strcmpi( info[index].name, name ) )
1151:         {
1152:             *bank = info[index].symbol;
1153:             return SUCCESS;
1154:         }
1155:         index++;
1156:     }
1157:     return -EC_BANK;
1158: }
1159:
1160: static int IDO_Bank_Name_To_Symbol( int board_id, const char * name, IDO_BANK_ENUM *
bank )
1161: {
1162:     int index;
1163:     const struct bank_info * info = board_definition[board_id].bank_info;
1164:     index = 0;
1165:     while ( NULL != info[index].name )
1166:     {
1167:         if ( 0 == strcmpi( info[index].name, name ) )
```

```
1168: {
1169:     *bank = info[index].symbol;
1170:     return SUCCESS;
1171: }
1172: index++;
1173: }
1174: return -EC_BANK;
1175: }
1176:
1177: static int Bank_Name_To_Symbol( int board_id, const char * name, int * bank )
1178: {
1179:     switch ( board_id )
1180:     {
1181:         case ID_IDI48: return IDI_Bank_Name_To_Symbol( board_id, name, (IDI_BANK_ENUM *) bank );
1182:         case ID_IDO48: return IDO_Bank_Name_To_Symbol( board_id, name, (IDO_BANK_ENUM *) bank );
1183:     }
1184:     return -EC_BOARD;
1185: }
1186:
1187:
1192: static int IDI_Register_Report_CSV( const struct idi_reg_definition * table, FILE * out )
1193: {
1194:     int index = 0;
1195:
1196:     fprintf( out, "\"acronym\",\"symbol\",\"bank\",\"direction\",\"physical_offset\"\n" );
1197:     do
1198:     {
1199:         fprintf( out, "\"%s\"", table[index].acronym );
1200:         fprintf( out, "\"%s\"", table[index].symbol_name );
1201:
1202:         switch( table[index].bank )
1203:         {
1204:             case IDI_BANK_0:   fprintf( out, "IDI_BANK_0" ); break;
1205:             case IDI_BANK_1:   fprintf( out, "IDI_BANK_1" ); break;
1206:             case IDI_BANK_2:   fprintf( out, "IDI_BANK_2" ); break;
1207:             case IDI_BANK_3:   fprintf( out, "IDI_BANK_3" ); break;
1208:             case IDI_BANK_4:   fprintf( out, "IDI_BANK_4" ); break;
1209:             case IDI_BANK_5:   fprintf( out, "IDI_BANK_5" ); break;
1210:             case IDI_BANK_6:   fprintf( out, "IDI_BANK_6" ); break;
1211:             case IDI_BANK_7:   fprintf( out, "IDI_BANK_7" ); break;
1212:             case IDI_BANK_NONE:   fprintf( out, "IDI_BANK_NONE" ); break;
1213:             case IDI_BANK_UNDEFINED:   fprintf( out, "IDI_BANK_UNDEFINED" ); break;
1214:         }
1215:         fprintf( out, "," );
1216:
1217:         switch( table[index].direction )
1218:         {
1219:             case REG_DIR_NONE:   fprintf( out, "REG_DIR_NONE" ); break;
1220:             case REG_DIR_READ:   fprintf( out, "REG_DIR_READ" ); break;
1221:             case REG_DIR_WRITE:   fprintf( out, "REG_DIR_WRITE" ); break;
1222:             case REG_DIR_READ_WRITE:   fprintf( out, "REG_DIR_READ_WRITE" ); break;
1223:         }
1224:         fprintf( out, "," );
1225:
1226:         fprintf( out, "\"%d\"", table[index].physical_offset );
1227:         fprintf( out, "\n\r" );
1228:         index++;
1229:     } while ( table[index].direction != REG_DIR_NONE );
1230:
1231:     return SUCCESS;
1232: }
```

```

1233:
1238: static int IDO_Register_Report_CSV( const struct ido_reg_definition * table, FILE * out
)
1239: {
1240:     int index = 0;
1241:
1242:     fprintf( out, "\"acronym\",\"symbol\",\"bank\",\"direction\",\"physical_offset\"\n" );
1243:     do
1244:     {
1245:         fprintf( out, "\"%s\",", table[index].acronym );
1246:         fprintf( out, "\"%s\",", table[index].symbol_name );
1247:
1248:         switch( table[index].bank )
1249:         {
1250:             case IDO_BANK_0:   fprintf( out, "IDO_BANK_0" ); break;
1251:             case IDO_BANK_1:   fprintf( out, "IDO_BANK_1" ); break;
1252:             case IDO_BANK_2:   fprintf( out, "IDO_BANK_2" ); break;
1253:             case IDO_BANK_3:   fprintf( out, "IDO_BANK_3" ); break;
1254:             case IDO_BANK_4:   fprintf( out, "IDO_BANK_4" ); break;
1255:             case IDO_BANK_5:   fprintf( out, "IDO_BANK_5" ); break;
1256:             case IDO_BANK_6:   fprintf( out, "IDO_BANK_6" ); break;
1257:             case IDO_BANK_7:   fprintf( out, "IDO_BANK_7" ); break;
1258:             case IDO_BANK_NONE: fprintf( out, "IDO_BANK_NONE" ); break;
1259:             case IDO_BANK_UNDEFINED: fprintf( out, "IDO_BANK_UNDEFINED" ); break;
1260:         }
1261:         fprintf( out, "," );
1262:
1263:         switch( table[index].direction )
1264:         {
1265:             case REG_DIR_NONE:   fprintf( out, "REG_DIR_NONE" ); break;
1266:             case REG_DIR_READ:   fprintf( out, "REG_DIR_READ" ); break;
1267:             case REG_DIR_WRITE:   fprintf( out, "REG_DIR_WRITE" ); break;
1268:             case REG_DIR_READ_WRITE: fprintf( out, "REG_DIR_READ_WRITE" ); break;
1269:         }
1270:         fprintf( out, "," );
1271:
1272:         fprintf( out, "\"%d\"", table[index].physical_offset );
1273:         fprintf( out, "\n\r" );
1274:         index++;
1275:     } while ( table[index].direction != REG_DIR_NONE );
1276:
1277:     return SUCCESS;
1278: }
1279:
1289: int Register_Report_CSV( int board_id, FILE * out )
1290: {
1291:     switch ( board_id )
1292:     {
1293:         case ID_IDI48: return IDI_Register_Report_CSV( ((const struct idi_reg_definition *) board_definition[board_id].definition), out );
1294:         case ID_IDO48: return IDO_Register_Report_CSV( ((const struct ido_reg_definition *) board_definition[board_id].definition), out );
1295:     }
1296:     return -EC_BOARD;
1297: }
1298:
1299:
1302: static int Register_Acronym_To_Row( int board_id, const char * acronym, int * row )
1303: {
1304:     int index;
1305:     const struct reg_definition * definition = (const struct reg_definition *) board_definition[board_id].definition;
1306:
1307:     index = 0;

```

```
1308: while ( definition[index].direction != REG_DIR_NONE )  
1309: {  
1310:     if ( 0 == strcasecmp( definition[index].acronym, acronym ) )  
1311:     {  
1312:         *row = index;  
1313:         return SUCCESS;  
1314:     }  
1315:     index++;  
1316: }  
1317: return -EC_NOT_FOUND;  
1318: }  
1319:  
1320:  
1321:  
1322:  
1323:  
1324:  
1325:  
1326:  
1334: BOOL String_To_Bool( const char * str )  
1335: {  
1336:     switch( str[0] )  
1337:     {  
1338:         case '0':  
1339:         case 'f':  
1340:         case 'F':  
1341:             return false;  
1342:         case '1':  
1343:         case 't':  
1344:         case 'T':  
1345:             return true;  
1346:     }  
1347:     return false;  
1348: }  
1349:  
1362: enum { HEX_DUMP_BYTES_PER_LINE = 16 };  
1363:  
1364: int Hex_Dump_Line( uint16_t address, size_t count, uint8_t * buffer, FILE * out )  
1365: {  
1366:     size_t index;  
1367:     char str_temp[8];  
1368:     char str_ascii[20];  
1369:     char str_hex_list[64];  
1370:  
1371:  
1372:     if ( count > HEX_DUMP_BYTES_PER_LINE ) return -EC_HEX_DUMP_COUNT;  
1373:  
1374:     sprintf( str_hex_list, "%04X: ", ((int) address) );  
1375:     strcpy( str_ascii, " " );  
1376:     for ( index = 0; index < count; index++ )  
1377:     {  
1378:         sprintf( str_temp, "%02X", buffer[index] );  
1379:         strcat( str_hex_list, str_temp );  
1380:  
1381:         if ( 0x07 == ( index & 0x07 ) ) strcat( str_hex_list, " " );  
1382:  
1383:         strcat( str_hex_list, " " );  
1384:  
1385:         if ( ( buffer[index] < ' ' ) || ( buffer[index] > '~' ) )  
1386:         {  
1387:             strcat( str_ascii, "." );  
1388:         }  
1389:     else  
1390:     {
```

```
1391:     sprintf( str_temp, "%c", buffer[index] );
1392:     strcat( str_ascii, str_temp );
1393: }
1394: }
1395:
1396: index = strlen( str_hex_list );
1397:
1398: count = 60 - index;
1399: while ( count > 0 )
1400: {
1401:     strcat( str_hex_list, " " );
1402:     count--;
1403: }
1404:
1405: fprintf( out, "%s%s\n", str_hex_list, str_ascii );
1406: return SUCCESS;
1407: }
1408:
1419: static const char ec_unknown[ ] = "unknown error code";
1420:
1421: const char * EC_Code_To_Human_Readable( EC_ENUM error_code )
1422: {
1423:     int index;
1424:
1425:     if ( error_code < 0 ) error_code = -error_code;
1426:
1427:     index = 0;
1428:     while( NULL != ec_human_readable[index].message )
1429:     {
1430:         if ( ((EC_ENUM) error_code) == ec_human_readable[index].error_code )
1431:         {
1432:             return ec_human_readable[index].message;
1433:         }
1434:         index++;
1435:     }
1436:     return ec_unknown;
1437: }
1438:
1450: BOOL Character_Get( int * character )
1451: {
1452:     int char_temp;
1453:     BOOL result = false;
1454: #ifdef __BORLANDC__
1455:     if ( kbhit() )
1456:     {
1457:         char_temp = getch();
1458:         result    = true;
1459:     }
1460:     else
1461:     {
1462:         char_temp = '\0';
1463:     }
1464: #else
1465:     char_temp = '\0';
1466:     result    = false;
1467: #endif
1468:     if ( NULL != character ) *character = char_temp;
1469:     return result;
1470: }
1471:
1483: int Time_Current_String( char * buf, size_t buf_size )
1484: {
1485:     struct tm * time_block;
1486:     time_t time_current;
```

```
1487: char *      time_ascii;
1488: size_t   index;
1489:
1490: time_current = time( NULL );
1491:
1492: time_block    = localtime( &time_current );
1493:
1497: time_ascii = asctime( time_block );
1498: index = 0;
1499:
1500: while( ( time_ascii[index] != '\n' ) && ( index < buf_size ) )
1501: {
1502:     buf[index] = time_ascii[index];
1503:     index++;
1504: }
1505: if ( index >= buf_size ) index = buf_size - 1;
1506: buf[index] = '\0';
1507: return SUCCESS;
1508: }
1509:
1512: static int Print_Multiple( char * message, FILE * out )
1513: {
1514:     int error_code;
1515:
1516:     error_code = fprintf( stdout, message );
1517:     if ( out != stdout )
1518:     {
1519:         error_code = fprintf( out, message );
1520:     }
1521:     return error_code;
1522: }
1523:
1524:
1525:
1526:
1527:
1528:
1529:
1530:
1531:
1535: int IO_Trace_Initialize( int board_id )
1536: {
1537:     size_t   index;
1538:     struct io_trace *  trace;
1539:     struct board_dataset * dataset;
1540:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
1541:
1542:     dataset->trace.begin = 0;
1543:     dataset->trace.end   = IO_TRACE_SIZE - 1;
1544:     dataset->trace.index = 0;
1545:     dataset->trace.count = 0;
1546:     dataset->trace.wrap  = false;
1547:     dataset->trace.posting = 0;
1548:
1549:     if ( IO_TRACE_START_ON == dataset->trace.start ) dataset->trace.active = true;
1550:     else                                         dataset->trace.active = false;
1551:
1552:     dataset->trace.buf = &(global_io_trace_buf[0]);
1553:
1554:     for ( index = 0; index < IO_TRACE_SIZE; index++ )
1555:     {
1556:         trace = &(dataset->trace.buf[index]);
1557:         memset( trace, 0, sizeof( struct io_trace ) );
1558:         dataset->trace.buf[index].action = IO_TRACE_NULL;
```

```
1559:     dataset->trace.buf[index].function_name = NULL;
1560:     dataset->trace.buf[index].board_id      = ID_BOARD_UNKNOWN;
1561: }
1562:
1563: if ( strlen(dataset->trace.filename) == 0 )
1564: {
1565:     strcpy( dataset->trace.filename, IO_TRACE_DUMP_FILE_NAME );
1566: }
1567: return SUCCESS;
1568: }
1569:
1570:
1571:
1575: static int IO_Trace_Log_Dataset_Begin( struct board_dataset * ds, size_t line, enum
IO_TRACE_ENUM action, int location, int error_code, uint8_t value )
1576: {
1577:     (void) line;
1578:     (void) location;
1579:     (void) value;
1580:
1581: if ( false == global_io_trace_enable ) return 1;
1582:
1583:
1584: switch( ds->trace.start )
1585: {
1586:     IO_TRACE_START_DEFINITION( IO_TRACE_START_STOP_EXTRACT_SWITCH_STATEMENT )
1587: }
1588:
1589: if ( false == ds->trace.active ) return 2;
1590:
1591:
1592: if ( false == ds->trace.wrap ) ds->trace.count++;
1593:
1594: if ( ds->trace.index >= ds->trace.end )
1595: {
1596:     ds->trace.index = ds->trace.begin;
1597:     ds->trace.wrap  = true;
1598: }
1599: else
1600: {
1601:     ds->trace.index++;
1602: }
1603: return 0;
1604: }
1605:
1609: static int IO_Trace_Log_Dataset_End( struct board_dataset * ds, size_t line, enum
IO_TRACE_ENUM action, int location, int error_code, uint8_t value )
1610: {
1611:     (void) line;
1612:     (void) action;
1613:     (void) location;
1614:     (void) value;
1615:
1616:
1617: switch( ds->trace.stop )
1618: {
1619:     IO_TRACE_STOP_DEFINITION( IO_TRACE_START_STOP_EXTRACT_SWITCH_STATEMENT )
1620: }
1621: return SUCCESS;
1622: }
1623:
1627: int IO_Trace_Log_Dataset( struct board_dataset * ds, size_t line, enum IO_TRACE_ENUM
action, int location, int error_code, uint8_t value )
1628: {
```

```
1629: struct io_trace * trace;
1630:
1631: if ( 0 != IO_Trace_Log_Dataset_Begin( ds, line, action, location, error_code, value ) )
1632: ) return SUCCESS;
1633:
1634:
1635:
1636:
1637:
1638:
1639:
1640:
1641:
1642:
1643:
1644:
1645:
1646:
1647:
1648:
1649:
1650:
1651:
1652:
1653:
1654:
1655:
1656:
1657: trace = &(ds->trace.buf[ds->trace.index]);
1658: trace->action = action;
1659: trace->board_id = ds->board_id;
1660: trace->location = location;
1661: trace->error_code = error_code;
1662: trace->value = value;
1663: trace->posting = ds->trace.posting;
1664: trace->line = line;
1665: ds->trace.posting++;
1666:
1667: IO_Trace_Log_Dataset_End( ds, line, action, location, error_code, value );
1668:
1669:
1670:
1671:
1672:
1673:
1674:
1675: return SUCCESS;
1676: }
1677:
1681: int IO_Trace_Log( int board_id, size_t line, enum IO_TRACE_ENUM action, int location,
int error_code, uint8_t value )
1682: {
1683: int ec;
1684: if ( true == global_io_trace_enable )
1685: {
1686: struct board_dataset * dataset;
1687: dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
1688:
1689: ec = IO_Trace_Log_Dataset( dataset, line, action, location, error_code, value );
1690: }
1691: else
1692: {
1693: ec = SUCCESS;
```

```

1694: }
1695: return ec;
1696: }
1697:
1701: int IO_Trace_Log_Function( const char * function_name, size_t line, enum IO_TRACE_ENUM
action, int board_id, int error_code )
1702: {
1703: int ec;
1704: if ( true == global_io_trace_enable )
1705: {
1706: struct io_trace * trace;
1707: struct board_dataset * ds;
1708: ds = ( struct board_dataset * ) board_definition[board_id].dataset;
1709:
1710: if ( 0 != IO_Trace_Log_Dataset_Begin( ds, line, action, 0, error_code, 0 ) ) return
SUCCESS;
1711:
1712: trace = &(ds->trace.buf[ds->trace.index]);
1713: trace->action = action;
1714: trace->board_id = ds->board_id;
1715: trace->location = 0;
1716: trace->error_code = error_code;
1717: trace->value = 0;
1718: trace->posting = ds->trace.posting;
1719: trace->line = line;
1720: trace->function_name = function_name;
1721: ds->trace.posting++;
1722:
1723: IO_Trace_Log_Dataset_End( ds, line, action, 0, error_code, 0 );
1724: }
1725: else
1726: {
1727: ec = SUCCESS;
1728: }
1729: return ec;
1730: }
1731:
1736: int IO_Trace_Dump( int board_id )
1737: {
1738: int error_code;
1739: FILE * out;
1740: size_t count, index, row;
1741: struct io_trace * trace;
1742: struct reg_definition * definition = ( struct reg_definition * )
board_definition[board_id].definition;
1743: struct board_dataset * dataset = ( struct board_dataset * )
board_definition[board_id].dataset;
1744:
1745: if ( false == global_io_trace_enable ) return SUCCESS;
1746:
1747: if ( strlen(dataset->trace.filename) == 0 ) out = fopen( IO_TRACE_DUMP_FILE_NAME, "wt"
);
1748: else out = fopen( dataset->trace.filename, "wt" );
1749:
1750: error_code = -EC_FILE_ERROR;
1751: if ( NULL == out ) return error_code;
1752: else error_code = SUCCESS;
1753:
1754: index = dataset->trace.index;
1755:
1756:
1757:
1758: count = dataset->trace.count;
1759:
```

```
1760:  
1761:  
1762:  
1763:  
1764: {  
1765:     char buf[64]={ 0 };  
1766:     Time_Current_String( buf, 64 );  
1767:     fprintf( out, "%s\n", buf );  
1768: }  
1769: fprintf( out, "trace.start      = " );  
1770: fprintf( out, "%s\n", global_io_trace_start_name[dataset->trace.start] );  
1771: fprintf( out, "trace.stop       = " );  
1772: fprintf( out, "%s\n", global_io_trace_stop_name[dataset->trace.stop] );  
1773: fprintf( out, "trace.file_name = " );  
1774: if ( 0 == strlen( dataset->trace.filename ) ) fprintf( out, "%s\n",  
IO_TRACE_DUMP_FILE_NAME );  
1775: else                                              fprintf( out, "%s\n",  
dataset->trace.filename );  
1776:  
1777:  
1778:  
1779: fprintf( out, "\\"row\\"\n" );  
1780:     fprintf( out, ", \\"posting\\"\n" );  
1781:     fprintf( out, ", \\"board\\"\n" );  
1782:     fprintf( out, ", \\"action\\"\n" );  
1783:     fprintf( out, ", \\"src_line\\"\n" );  
1784:     fprintf( out, ", \\"register_or_function\\"\n" );  
1785:     fprintf( out, ", \\"channel\\"\n" );  
1786:     fprintf( out, ", \\"error\\"\n" );  
1787:     fprintf( out, ", \\"value\\"\n" );  
1788:     fprintf( out, "\n" );  
1789:  
1790: row = 0;  
1791: while ( count > 0 )  
1792: {  
1793:     trace = &(dataset->trace.buf[index]);  
1794:  
1795:  
1796:     fprintf( out, "%3u", (unsigned) row );  
1797:  
1798:  
1799:     fprintf( out, ", %6u", (unsigned) trace->posting );  
1800:  
1801:  
1802:  
1803:     fprintf( out, ", " );  
1804:     switch( trace->board_id )  
1805:     {  
1806:         case ID_IDI48: fprintf( out, "\\"IDI48\\"\n" ); break;  
1807:         case ID_IDO48: fprintf( out, "\\"IDO48\\"\n" ); break;  
1808:         case ID_IAI16: fprintf( out, "\\"IAI16\\"\n" ); break;  
1809:         case ID_ARM32: fprintf( out, "\\"ARM32\\"\n" ); break;  
1810:     }  
1811:  
1812:     fprintf( out, ", " );  
1813:     fprintf( out, "\\"%s\\"", global_io_trace_name[trace->action] );  
1814:  
1815:  
1816:     fprintf( out, ", %6u", (unsigned) trace->line );  
1817:  
1818:  
1819:     switch( trace->action )  
1820:     {  
1821:         case IO_TRACE_READ:
```

```

1822:     case IO_TRACE_WRITE:
1823:         fprintf( out, ", \"reg: %s\"", definition[REG_LOCATION_LOGICAL_GET(trace->location)
+ REG_LOCATION_CHANNEL_GET( trace->location )].symbol_name );
1824:         break;
1825:     default:
1826:         if ( NULL != trace->function_name )
1827:         {
1828:             fprintf( out, ", \"fnc: %s\"", trace->function_name );
1829:         }
1830:         break;
1831:     }
1832:
1833:
1834:     fprintf( out, ", %d", REG_LOCATION_CHANNEL_GET( trace->location ) );
1835:
1836:     fprintf( out, ", \"%s\"", EC_Code_To_Human_Readable( trace->error_code ) );
1837:
1838:     fprintf( out, ", 0x%02X", (int) trace->value );
1839:
1840:     fprintf( out, "\n" );
1841:
1842:
1843:     if ( index == dataset->trace.begin )
1844:     {
1845:         index = dataset->trace.end;
1846:     }
1847:     else
1848:     {
1849:         index--;
1850:     }
1851:     count--;
1852:     row++;
1853: }
1854:
1855: fclose( out );
1856: return error_code;
1857: }
1858:
1863: #define IDI_IO_DIRECTION_TEST    1
1864:
1865: #if(0)
1866:
1875: static char * IO_Get_Symbol_Name( IDI_REG_ENUM location )
1876: {
1877:     int index;
1878:     index = REG_LOCATION_LOGICAL_GET( location );
1879:     return idi_definitions[index].symbol_name;
1880: }
1881:
1892: #if defined( IDI_IO_DIRECTION_TEST )
1893: static BOOL IO_Direction_IsNotValid( IDI_REG_ENUM location, REG_DIR_ENUM direction )
1894: {
1895:     int index;
1896:     index = REG_LOCATION_LOGICAL_GET( location );
1897:     if ( direction == (idi_definitions[index].direction & direction) ) return false;
1898:     return true;
1899: }
1900: #endif
1901: #endif
1902:
1903:
1904:
1907: static int IO_Write_U8_Port( struct board_dataset * dataset, size_t line, int address,
uint8_t value )

```

```
1908: {
1909:     if ( NULL == dataset )
1910:     {
1911: #if defined( __MSDOS__ )
1912:         outportb( address, value );
1913: #else
1914:         printf( "IO_Write_U8_Port: address = 0x%04X, value = 0x%02X\n", address, value );
1915: #endif
1916:     }
1917:     else
1918:     {
1919:         if ( !dataset->io_simulate )
1920:         {
1921: #if defined( __MSDOS__ )
1922:             outportb( address, value );
1923: #endif
1924:         }
1925:         if ( ( dataset->io_report ) || ( dataset->io_simulate ) )
1926:         {
1927:             printf( "IO_Write_U8_Port: address = 0x%04X, value = 0x%02X\n", address, value );
1928:         }
1929:     }
1930:     return SUCCESS;
1931: }
1932:
1933: static int IO_Read_U8_Port( struct board_dataset * dataset, size_t line, int address,
1934: uint8_t * value )
1935: {
1936:     if ( NULL == dataset )
1937:     {
1938: #if defined( __MSDOS__ )
1939:         *value = inportb( address );
1940: #else
1941:         printf( "IO_Read_U8_Port: address = 0x%04X, value=unknown\n", address );
1942: #endif
1943:     }
1944:     else
1945:     {
1946:         if ( !dataset->io_simulate )
1947:         {
1948: #if defined( __MSDOS__ )
1949:             *value = inportb( address );
1950: #endif
1951: #endif
1952:     }
1953:     if ( ( dataset->io_report ) || ( dataset->io_simulate ) )
1954:     {
1955:         printf( "IO_Read_U8_Port: address = 0x%04X, ", address );
1956: #if defined( __MSDOS__ )
1957:         printf( "value = 0x%02X\n", *value );
1958: #else
1959:         printf( "value = unknown\n" );
1960: #endif
1961:     }
1962: }
1963: return SUCCESS;
1964: }
1965:
1966:
1979: int IO_Write_U8( int board_id, size_t line, int location, uint8_t value )
1980: {
1981:     int error_code;
1982:     int index;
1983:     int offset;
1984:     int channel;
```

```

1985: int address;
1986: int bank;
1987:
1988: struct reg_definition * definition = ( struct reg_definition * )
board_definition[board_id].definition;
1989: struct board_dataset * dataset = ( struct board_dataset * )
board_definition[board_id].dataset;
1990:
1991: error_code = SUCCESS;
1992: index = REG_LOCATION_LOGICAL_GET( location );
1993: #if defined( IDI_IO_DIRECTION_TEST )
1994: if ( !( REG_DIR_WRITE == ( definition[index].direction & REG_DIR_WRITE ) ) )
1995: {
1996: printf( "IO_Write_U8: %s, error in direction\n", definition[index].symbol_name );
1997: error_code = -EC_DIRECTION;
1998: goto IO_Write_U8_Exit;
1999: }
2000: #endif
2001: bank = definition[index].bank;
2002: if ( ( BANK_NONE != bank ) && ( bank != dataset->bank_previous ) )
2003: {
2004: address = dataset->base_address + definition[REG_LOCATION_LOGICAL_GET(
board_definition[board_id].bank_register_symbol)].physical_offset;
2005: dataset->bank_previous = bank;
2006: IO_Write_U8_Port( dataset, __LINE__, address, (uint8_t) bank );
2007: IO_Trace_Log( board_id, line, IO_TRACE_WRITE,
board_definition[board_id].bank_register_symbol, error_code, (uint8_t) bank );
2008: if ( ( dataset->io_report ) || ( dataset->io_simulate ) )
2009: {
2010: printf( "IO_Write_U8 bank write: %s, address = 0x%04X, bank = %s (0x%02X)\n",
definition[index].symbol_name,
2011: address,
2012: Bank_Symbol_To_Name( board_id, bank ),
2013: bank
2014: );
2015: }
2016: }
2017: channel = REG_LOCATION_CHANNEL_GET( location );
2018: offset = definition[index].physical_offset;
2019: address = dataset->base_address + offset + channel;
2020: IO_Write_U8_Port( dataset, __LINE__, address, value );
2021: IO_Write_U8_Exit:
2022: IO_Trace_Log( board_id, line, IO_TRACE_WRITE, location, error_code, value );
2023: return SUCCESS;
2024: }
2025:
2026:
2038: int IO_Read_U8( int board_id, size_t line, int location, uint8_t * value )
2039: {
2040: int error_code;
2041: int index;
2042: int offset;
2043: int channel;
2044: int address;
2045: int bank;
2046:
2047: struct reg_definition * definition = ( struct reg_definition * )
board_definition[board_id].definition;
2048: struct board_dataset * dataset = ( struct board_dataset * )
board_definition[board_id].dataset;
2049:
2050: error_code = SUCCESS;
2051: index = REG_LOCATION_LOGICAL_GET( location );
2052: #if defined( IDI_IO_DIRECTION_TEST )

```

```
2053: if ( !( REG_DIR_READ == ( definition[index].direction & REG_DIR_READ ) ) )
2054: {
2055:     printf( "IO_Read_U8: %s, error in direction\n", definition[index].symbol_name );
2056:     error_code = -EC_DIRECTION;
2057:     goto IO_Read_U8_Exit;
2058: }
2059: #endif
2060: bank    = definition[index].bank;
2061: if ( ( BANK_NONE != bank ) && ( bank != dataset->bank_previous ) )
2062: {
2063:     address = dataset->base_address + definition[REG_LOCATION_LOGICAL_GET(
board_definition[board_id].bank_register_symbol)].physical_offset;
2064:     dataset->bank_previous = bank;
2065:     IO_Write_U8_Port( dataset, __LINE__, address, (uint8_t) bank );
2066:     IO_Trace_Log( board_id, line, IO_TRACE_READ,
board_definition[board_id].bank_register_symbol, error_code, (uint8_t) bank );
2067:     if ( ( dataset->io_report ) || ( dataset->io_simulate ) )
2068:     {
2069:         printf( "IO_Read_U8 bank write: %s, address = 0x%04X, bank = %s (0x%02X)\n",
definition[index].symbol_name,
2070:                         address,
2071:                         Bank_Symbol_To_Name( board_id, bank ),
2072:                         bank
2073:                 );
2074:     }
2075: }
2076: channel = REG_LOCATION_CHANNEL_GET( location );
2077: offset   = definition[index].physical_offset;
2078: address  = dataset->base_address + offset + channel;
2079: IO_Read_U8_Port( dataset, __LINE__, address, value );
2080: IO_Read_U8_Exit:
2081: IO_Trace_Log( board_id, line, IO_TRACE_READ, location, error_code, *value );
2082: return SUCCESS;
2083: }
2084:
2099: void IO_Write_U16_Address_Increment( int board_id, size_t line, int location, uint16_t
value )
2100: {
2101:
2102:
2103:
2104:     IO_Write_U8( board_id, line, (int)((int) location + 0 ), (uint8_t)( value      &
0xFF ) );
2105:     IO_Write_U8( board_id, line, (int)((int) location + 1 ), (uint8_t)( ( value >> 8 ) &
0xFF ) );
2106: }
2107:
2122: void IO_Read_U16_Address_Increment( int board_id, size_t line, int location, uint16_t
* value )
2123: {
2124:
2125:     uint8_t lsb, msb;
2126:     IO_Read_U8( board_id, line, (int)((int) location + 0 ), &lsb );
2127:     IO_Read_U8( board_id, line, (int)((int) location + 1 ), &msb );
2128:     *value = ( ((uint16_t) msb) << 8 ) | ( ((uint16_t) lsb) & 0xFF );
2129: }
2130:
2145: void IO_Write_U16_Address_Fixed( int board_id, size_t line, int location, uint16_t
value )
2146: {
2147:
2148:     IO_Write_U8( board_id, line, (int)((int) location + 0 ), (uint8_t)( value      &
0xFF ) );
2149:     IO_Write_U8( board_id, line, (int)((int) location + 0 ), (uint8_t)( ( value >> 8 ) &
```

```
0xFF ) );
2150: }
2151:
2166: void IO_Read_U16_Address_Fixed( int board_id, size_t line, int location, uint16_t * value )
2167: {
2168:
2169:     uint8_t lsb, msb;
2170:     IO_Read_U8( board_id, line, (int)((int) location + 0), &lsb );
2171:     IO_Read_U8( board_id, line, (int)((int) location + 0), &msb );
2172:     *value = ( ((uint16_t) msb) << 8 ) | ( ((uint16_t) lsb) & 0xFF );
2173: }
2174:
2175:
2176:
2177:
2178:
2179:
2180:
2189:
2190:
2191: #if defined( __MSDOS__ )
2192:
2193: #include <limits.h>
2194:
2195: #define TARGET_CPU_INTEL_386 1
2196:
2197:
2198:
2201: #define IOKERN_DOS_PIC1_BASE_ADDRESS 0x20
2202: #define IOKERN_DOS_PIC2_BASE_ADDRESS 0xA0
2203: #define IOKERN_DOS_PIC1_CMD           IOKERN_DOS_PIC1_BASE_ADDRESS
2204: #define IOKERN_DOS_PIC1_IMR          ( IOKERN_DOS_PIC1_BASE_ADDRESS + 1 )
2205: #define IOKERN_DOS_PIC2_CMD          IOKERN_DOS_PIC2_BASE_ADDRESS
2206: #define IOKERN_DOS_PIC2_IMR          ( IOKERN_DOS_PIC2_BASE_ADDRESS + 1 )
2207:
2208:
2210: #define IOKERN_DOS_NSEOI           0x20
2211:
2212:
2213: #define IOKERN_IRQ_START disable()
2214: #define IOKERN_IRQ_ENABLE enable()
2215: #define IOKERN_IRQ_END(irq) IOKern_PIC_EOI( irq ); enable()
2216:
2217: #if defined( TARGET_CPU_INTEL_186 )
2218: # pragma message "TARGET_CPU_INTEL_186 for interrupt chaining"
2219:
2220: struct pt_regs
2221: {
2222:     uint16_t bp;
2223:     uint16_t di;
2224:     uint16_t si;
2225:     uint16_t ds;
2226:     uint16_t es;
2227:     uint16_t dx;
2228:     uint16_t cx;
2229:     uint16_t bx;
2230:     uint16_t ax;
2231:     uint16_t ip;
2232:     uint16_t cs;
2233:     uint16_t flags;
2234: };
2235:
2236:
```

```
2237: # define IOKERN_IRQ_CHAIN_SELECT(old_fp,regs) \
2238:   _BX = regs.bx; \
2239:   _CX = regs.ax; \
2240:   regs.ax = FP_SEG((void far *)old_fp); \
2241:   regs.bx = FP_OFF((void far *)old_fp); \
2242:   _AX = _CX ; \
2243:   __emit__(0x5D); \
2244:   __emit__(0x5F); \
2245:   __emit__(0x5E); \
2246:   __emit__(0x1F); \
2247:   __emit__(0x07); \
2248:   __emit__(0x5A); \
2249:   __emit__(0x59); \
2250:   __emit__(0xCB);
2251:
2252: #elif defined( TARGET_CPU_INTEL_386 )
2253: # pragma message "TARGET_CPU_INTEL_386 for interrupt chaining"
2254:
2255: struct pt_regs
2256: {
2257:   uint16_t bp;
2258:   union { uint32_t edi; uint16_t di; };
2259:   union { uint32_t esi; uint16_t si; };
2260:   uint16_t ds;
2261:   uint16_t es;
2262:   union { uint32_t edx; uint16_t dx; struct { uint8_t l, h; } d; };
2263:   union { uint32_t ecx; uint16_t cx; struct { uint8_t l, h; } c; };
2264:   union { uint32_t ebx; uint16_t bx; struct { uint8_t l, h; } b; };
2265:
2266:   union { uint32_t eax; uint16_t ax; struct { uint16_t l, h; } aw; struct { uint8_t l,
2267: h; } a; };
2268:   uint16_t ip;
2269:   uint16_t cs;
2270:   uint16_t flags;
2271: };
2272: # define IOKERN_IRQ_CHAIN_SELECT(old_fp,regs) \
2273:   _ECX = regs.eax; \
2274:   regs.aw.h = FP_SEG((void far *)old_fp); \
2275:   regs.aw.l = FP_OFF((void far *)old_fp); \
2276:   _EAX = _ECX ; \
2277:   __emit__(0x5D); \
2278:   __emit__(0x66); __emit__(0x5F); \
2279:   __emit__(0x66); __emit__(0x5E); \
2280:   __emit__(0x1F); \
2281:   __emit__(0x07); \
2282:   __emit__(0x66); __emit__(0x5A); \
2283:   __emit__(0x66); __emit__(0x59); \
2284:   __emit__(0x66); __emit__(0x5B); \
2285:   __emit__(0xCB);
2286: #endif
2287:
2288: #define INTERRUPT interrupt
2289: typedef void INTERRUPT ( * IOKERN_ISR_FP )( struct pt_regs r );
2290:
2291:
2292:
2293: #define IOKERN_LOCAL_IRQ_SAVE(flags) \
2294:   do { \
2295:     asm push bx ; \
2296:     asm pushf ; \
2297:     asm pop bx ; \
2298:     asm mov [flags], bx ; \
2299:     asm cli ; \
```

```

2300:     asm pop bx    ; \
2301: } while(0)
2302:
2303: #define IOKERN_LOCAL_IRQ_RESTORE(flags) \
2304: do {           \
2305:     asm push bx    ; \
2306:     asm mov bx, [flags] ; \
2307:     asm push bx    ; \
2308:     asm popf    ; \
2309:     asm pop bx    ; \
2310: } while(0)
2311:
2312:
2313: typedef enum
2314: {
2315:     IOKERN_IRQ_NONE    = UINT_MAX,
2316:
2317:     IOKERN_IRQ_0    = 0,
2318:     IOKERN_IRQ_1    = 1,
2319:     IOKERN_IRQ_2    = 2,
2320:     IOKERN_IRQ_3    = 3,
2321:     IOKERN_IRQ_4    = 4,
2322:     IOKERN_IRQ_5    = 5,
2323:     IOKERN_IRQ_6    = 6,
2324:     IOKERN_IRQ_7    = 7,
2325:     IOKERN_IRQ_8    = 8,
2326:     IOKERN_IRQ_9    = 9,
2327:     IOKERN_IRQ_10   = 10,
2328:     IOKERN_IRQ_11   = 11,
2329:     IOKERN_IRQ_12   = 12,
2330:     IOKERN_IRQ_13   = 13,
2331:     IOKERN_IRQ_14   = 14,
2332:     IOKERN_IRQ_15   = 15,
2333:
2334:     IOKERN_INT_33   = 0x33
2335: } IOKERN_IRQ_ENUM;
2336:
2337: typedef enum
2338: {
2339:     IOKERN_TASK_ID_NONE    = -1,
2340:     IOKERN_TASK_ID_IRQ0   = 0,
2341:     IOKERN_TASK_ID_IRQ1   = 1,
2342:     IOKERN_TASK_ID_IRQ2   = 2,
2343:     IOKERN_TASK_ID_IRQ3   = 3,
2344:     IOKERN_TASK_ID_IRQ4   = 4,
2345:     IOKERN_TASK_ID_IRQ5   = 5,
2346:     IOKERN_TASK_ID_IRQ6   = 6,
2347:     IOKERN_TASK_ID_IRQ7   = 7,
2348:     IOKERN_TASK_ID_IRQ8   = 8,
2349:     IOKERN_TASK_ID_IRQ9   = 9,
2350:     IOKERN_TASK_ID_IRQ10  = 10,
2351:     IOKERN_TASK_ID_IRQ11  = 11,
2352:     IOKERN_TASK_ID_IRQ12  = 12,
2353:     IOKERN_TASK_ID_IRQ13  = 13,
2354:     IOKERN_TASK_ID_IRQ14  = 14,
2355:     IOKERN_TASK_ID_IRQ15  = 15,
2356:     IOKERN_TASK_ID_INT33  = 16
2357: } IOKERN_TASK_ID_ENUM;
2358:
2359:
2360: typedef enum
2361: {
2362:     IOKERN_CHAIN_TO_OLD_OFF  = 0,
2363:     IOKERN_CHAIN_TO_OLD_TIMER = 1,

```

```
2367: IOKERN_CHAIN_TO_OLD_NORMAL = 2
2368: } IOKERN_CHAIN_TO_OLD_ENUM;
2369:
2370:
2371:
2377:
2383: enum irqreturn {
2384:     IRQ_NONE,
2385:     IRQ_HANDLED,
2386:     IRQ_WAKE_THREAD,
2387: };
2388:
2389: typedef enum irqreturn irqreturn_t;
2390:
2391:
2392: typedef irqreturn_t (* IOKERN_TASK_FP)( int irq, void * dev_id, struct pt_regs * regs );
2393: typedef void          (* IOKERN_HELP_FP)( IOKERN_TASK_ID_ENUM id, IOKERN_IRQ_ENUM irq,
2394:                                         struct pt_regs * regs );
2394:
2395:
2396: typedef struct IOKERN_TASK_TYPE
2397: {
2398:
2399:     IOKERN_TASK_FP task_fp;
2400:     IOKERN_HELP_FP help_fp;
2401:     void * dev_id;
2402:     const char * name;
2403:     size_t speed;
2404:     IOKERN_CHAIN_TO_OLD_ENUM chain_to_old;
2405:     unsigned int number;
2406:     size_t sw_int;
2407:     irqreturn_t result;
2408:     unsigned long count;
2409:     IOKERN_ISR_FP old_isr;
2410:     unsigned char old_state;
2411: } IOKERN_TASK_TYPE;
2412:
2413:
2415: static void IO Kern_DOS_IRQ_Mask_Set( unsigned char irq, unsigned char state )
2416: {
2417:     unsigned int port;
2418:     unsigned char value;
2419:
2420:     port = IOKERN_DOS_PIC1_IMR;
2421:     if ( irq >= 8 )
2422:     {
2423:
2424:         if ( state ) value = inp( port ) | ( 1 << 2 );
2425:         else         value = inp( port ) & ~( 1 << 2 );
2426:         outp( port, value );
2427:
2428:         irq = irq - 8;
2429:         port = IOKERN_DOS_PIC2_IMR;
2430:     }
2431:     if ( state ) value = inp( port ) | ( 1 << irq );
2432:     else         value = inp( port ) & ~( 1 << irq );
2433:     outp( port, value );
2434: }
2435:
2437: static void IO Kern_DOS_IRQ_Mask_Get( unsigned char irq, unsigned char * state )
2438: {
2439:     unsigned int port;
2440:     unsigned char value;
```

```
2441: if ( irq < 8 )
2442: {
2443:     port = IOKERN_DOS_PIC1_IMR;
2444: }
2445: else
2446: {
2447:     irq = irq - 8;
2448:     port = IOKERN_DOS_PIC2_IMR;
2449: }
2450: value = inp( port ) & ( 1 << irq );
2451: *state = value;
2452: }
2453:
2454:
2455: typedef void interrupt (* IOKERN_DOS_VECTOR_TYPE )();
2456:
2457:
2458:
2459: IOKERN_DOS_VECTOR_TYPE IOKern_DOS_Vector_Get( unsigned num )
2460:
2461: {
2462:     asm push es
2463:     asm push bx
2464:
2465:     asm xor bx,bx
2466:     asm mov es,bx
2467:     asm mov bx,[num]
2468:
2469:     asm shl bx,1
2470:     asm shl bx,1
2471:
2472:     asm les bx,es:[bx]
2473:
2474:     asm mov dx,es
2475:     asm mov ax,bx
2476:     asm pop bx
2477:     asm pop es
2478:
2479: }
2480:
2481: }
2482:
2483:
2484: void IOKern_DOS_Vector_Set( unsigned num, IOKERN_DOS_VECTOR_TYPE h )
2485:
2486: {
2487:     asm push es
2488:     asm push bx
2489:
2490:     asm xor bx,bx
2491:     asm mov es,bx
2492:     asm mov bx,[num]
2493:
2494:     asm shl bx,1
2495:     asm shl bx,1
2496:
2497:     asm pushf
2498:     asm cli
2499:     asm mov ax,word ptr [h]
2500:     asm mov es:[bx],ax
2501:     asm mov es:[bx+2],ax
2502:     asm mov ax,word ptr [h+2]
2503:     asm mov es:[bx+2],ax
2504:     asm popf
2505:     asm pop bx
2506:     asm pop es
2507: }
2508:
2509: #define IOKERN_DOS_INT_GET(num) IOKern_DOS_Vector_Get(num)
2510: #define IOKERN_DOS_INT_SET(num, fn) IOKern_DOS_Vector_Set(num,
```

```
(IOKERN_DOS_VECTOR_TYPE)fn)
2512:
2513: #define IOKERN_TASK_QTY 17
2514: IOKERN_TASK_TYPE iokern_task[IOKERN_TASK_QTY];
2515:
2516:
2520: void IOKern_PIC_EOI( unsigned char irq )
2521: {
2522:   outp( IOKERN_DOS_PIC1_CMD, IOKERN_DOS_NSEOI );
2523:   if ( irq >= 8 ) outp( IOKERN_DOS_PIC2_CMD, IOKERN_DOS_NSEOI );
2524: }
2525:
2530: void IOKern Interrupt_Helper( IOKERN_TASK_ID_ENUM id, IOKERN_IRQ_ENUM irq, struct
pt_regs * regs )
2531: {
2532:   iokern_task[id].result =
2533:     ( * iokern_task[id].task_fp )( irq, iokern_task[id].dev_id, regs );
2534:
2535:   iokern_task[id].count++;
2536: }
2537:
2543: static void INTERRUPT IOKern_ISR0( struct pt_regs r )
2544: {
2545: #if defined( IOKERN_ISR_USE_THESE_GUTS )
2546:   IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ1, IOKERN_IRQ_0, r );
2547: #else
2548:   IOKERN_IRQ_START;
2549:   IOKern Interrupt_Helper( IOKERN_TASK_ID_IRQ0, IOKERN_IRQ_0, &r );
2550:   IOKERN_IRQ_END( IOKERN_IRQ_0 );
2551: #endif
2552: }
2553:
2559: static void INTERRUPT IOKern_ISR1( struct pt_regs r )
2560: {
2561: #if defined( IOKERN_ISR_USE_THESE_GUTS )
2562:   IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ1, IOKERN_IRQ_1, r );
2563: #else
2564:   IOKERN_IRQ_START;
2565:   IOKern Interrupt_Helper( IOKERN_TASK_ID_IRQ1, IOKERN_IRQ_1, &r );
2566:   IOKERN_IRQ_END( IOKERN_IRQ_1 );
2567: #endif
2568: }
2569:
2575: static void INTERRUPT IOKern_ISR2( struct pt_regs r )
2576: {
2577: #if defined( IOKERN_ISR_USE_THESE_GUTS )
2578:   IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ2, IOKERN_IRQ_2, r );
2579: #else
2580:   IOKERN_IRQ_START;
2581:   IOKern Interrupt_Helper( IOKERN_TASK_ID_IRQ2, IOKERN_IRQ_2, &r );
2582:   IOKERN_IRQ_END( IOKERN_IRQ_2 );
2583: #endif
2584: }
2585:
2591: static void INTERRUPT IOKern_ISR3( struct pt_regs r )
2592: {
2593: #if defined( IOKERN_ISR_USE_THESE_GUTS )
2594:   IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ3, IOKERN_IRQ_3, r );
2595: #else
2596:   IOKERN_IRQ_START;
2597:   IOKern Interrupt_Helper( IOKERN_TASK_ID_IRQ3, IOKERN_IRQ_3, &r );
2598:   IOKERN_IRQ_END( IOKERN_IRQ_3 );
2599: #endif
2600: }
```

```
2601:  
2602: static void INTERRUPT IOKern_ISR4( struct pt_regs r )  
2603: {  
2604: #if defined( IOKERN_ISR_USE_THESE_GUTS )  
2605: IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ4, IOKERN_IRQ_4, r );  
2606: #else  
2607: IOKERN_IRQ_START;  
2608: IOKern Interrupt_Helper( IOKERN_TASK_ID_IRQ4, IOKERN_IRQ_4, &r );  
2609: IOKERN_IRQ_END( IOKERN_IRQ_4 );  
2610: #endif  
2611: }  
2612:  
2613:  
2614:  
2615:  
2616:  
2617:  
2618:  
2619:  
2620:  
2621:  
2622:  
2623: static void INTERRUPT IOKern_ISR5( struct pt_regs r )  
2624: {  
2625: #if defined( IOKERN_ISR_USE_THESE_GUTS )  
2626: IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ5, IOKERN_IRQ_5, r );  
2627: #else  
2628: IOKERN_IRQ_START;  
2629: IOKern Interrupt_Helper( IOKERN_TASK_ID_IRQ5, IOKERN_IRQ_5, &r );  
2630: IOKERN_IRQ_END( IOKERN_IRQ_5 );  
2631: #endif  
2632: }  
2633:  
2634:  
2635: static void INTERRUPT IOKern_ISR6( struct pt_regs r )  
2636: {  
2637: #if defined( IOKERN_ISR_USE_THESE_GUTS )  
2638: IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ6, IOKERN_IRQ_6, r );  
2639: #else  
2640: IOKERN_IRQ_START;  
2641: IOKern Interrupt_Helper( IOKERN_TASK_ID_IRQ6, IOKERN_IRQ_6, &r );  
2642: IOKERN_IRQ_END( IOKERN_IRQ_6 );  
2643: #endif  
2644: }  
2645:  
2646:  
2647:  
2648:  
2649:  
2650:  
2651: static void INTERRUPT IOKern_ISR7( struct pt_regs r )  
2652: {  
2653: #if defined( IOKERN_ISR_USE_THESE_GUTS )  
2654: IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ7, IOKERN_IRQ_7, r );  
2655: #else  
2656: IOKERN_IRQ_START;  
2657: IOKern Interrupt_Helper( IOKERN_TASK_ID_IRQ7, IOKERN_IRQ_7, &r );  
2658: IOKERN_IRQ_END( IOKERN_IRQ_7 );  
2659: #endif  
2660: }  
2661:  
2662:  
2663:  
2664:  
2665:  
2666:  
2667: static void INTERRUPT IOKern_ISR8( struct pt_regs r )  
2668: {  
2669: #if defined( IOKERN_ISR_USE_THESE_GUTS )  
2670: IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ8, IOKERN_IRQ_8, r );  
2671: #else  
2672: IOKERN_IRQ_START;  
2673: IOKern Interrupt_Helper( IOKERN_TASK_ID_IRQ8, IOKERN_IRQ_8, &r );  
2674: IOKERN_IRQ_END( IOKERN_IRQ_8 );  
2675: #endif  
2676: }  
2677:  
2678:  
2679:  
2680:  
2681:  
2682:  
2683:  
2684:  
2685:  
2686:  
2687: static void INTERRUPT IOKern_ISR9( struct pt_regs r )  
2688: {  
2689: #if defined( IOKERN_ISR_USE_THESE_GUTS )  
2690: IOKERN_ISR_GUTS( IOKERN_TASK_ID_IRQ9, IOKERN_IRQ_9, r );  
2691: #else  
2692: IOKERN_IRQ_START;  
2693: IOKern Interrupt_Helper( IOKERN_TASK_ID_IRQ9, IOKERN_IRQ_9, &r );  
2694: IOKERN_IRQ_END( IOKERN_IRQ_9 );
```

```
2695: #endif
2696: }
2697:
2703: static void INTERRUPT IO Kern_ISR10( struct pt_regs r )
2704: {
2705: #if defined( IOKERN_ISR_USE THESE_GUTS )
2706: IO KERN_ISR_GUTS( IO KERN_TASK_ID_IRQ10, IO KERN_IRQ_10, r );
2707: #else
2708: IO KERN_IRQ_START;
2709: IO KERN Interrupt_Helper( IO KERN_TASK_ID_IRQ10, IO KERN_IRQ_10, &r );
2710: IO KERN IRQ_END( IO KERN_IRQ_10 );
2711: #endif
2712: }
2713:
2719: static void INTERRUPT IO Kern_ISR11( struct pt_regs r )
2720: {
2721: #if defined( IOKERN_ISR_USE THESE_GUTS )
2722: IO KERN_ISR_GUTS( IO KERN_TASK_ID_IRQ11, IO KERN_IRQ_11, r );
2723: #else
2724: IO KERN_IRQ_START;
2725: IO KERN Interrupt_Helper( IO KERN_TASK_ID_IRQ11, IO KERN_IRQ_11, &r );
2726: IO KERN IRQ_END( IO KERN_IRQ_11 );
2727: #endif
2728: }
2729:
2735: static void INTERRUPT IO Kern_ISR12( struct pt_regs r )
2736: {
2737: #if defined( IOKERN_ISR_USE THESE_GUTS )
2738: IO KERN_ISR_GUTS( IO KERN_TASK_ID_IRQ12, IO KERN_IRQ_12, r );
2739: #else
2740: IO KERN_IRQ_START;
2741: IO KERN Interrupt_Helper( IO KERN_TASK_ID_IRQ12, IO KERN_IRQ_12, &r );
2742: IO KERN IRQ_END( IO KERN_IRQ_12 );
2743: #endif
2744: }
2745:
2751: static void INTERRUPT IO Kern_ISR13( struct pt_regs r )
2752: {
2753: #if defined( IOKERN_ISR_USE THESE_GUTS )
2754: IO KERN_ISR_GUTS( IO KERN_TASK_ID_IRQ13, IO KERN_IRQ_13, r );
2755: #else
2756: IO KERN_IRQ_START;
2757: IO KERN Interrupt_Helper( IO KERN_TASK_ID_IRQ13, IO KERN_IRQ_13, &r );
2758: IO KERN IRQ_END( IO KERN_IRQ_13 );
2759: #endif
2760: }
2761:
2767: static void INTERRUPT IO Kern_ISR14( struct pt_regs r )
2768: {
2769: #if defined( IOKERN_ISR_USE THESE_GUTS )
2770: IO KERN_ISR_GUTS( IO KERN_TASK_ID_IRQ14, IO KERN_IRQ_14, r );
2771: #else
2772: IO KERN_IRQ_START;
2773: IO KERN Interrupt_Helper( IO KERN_TASK_ID_IRQ14, IO KERN_IRQ_14, &r );
2774: IO KERN IRQ_END( IO KERN_IRQ_14 );
2775: #endif
2776: }
2777:
2783: static void INTERRUPT IO Kern_ISR15( struct pt_regs r )
2784: {
2785: #if defined( IOKERN_ISR_USE THESE_GUTS )
2786: IO KERN_ISR_GUTS( IO KERN_TASK_ID_IRQ15, IO KERN_IRQ_15, r );
2787: #else
2788: IO KERN_IRQ_START;
```

```
2789: IOKern_Interrupt_Helper( IOKERN_TASK_ID_IRQ15, IOKERN_IRQ_15, &r );
2790: IOKERN_IRQ_END( IOKERN_IRQ_15 );
2791: #endif
2792: }
2793:
2794: IOKERN_ISR_FPs iokern_isr_table[ IOKERN_TASK_QTY ] =
2795: {
2796:     IOKern_ISR0,
2797:     IOKern_ISR1,
2798:     IOKern_ISR2,
2799:     IOKern_ISR3,
2800:     IOKern_ISR4,
2801:     IOKern_ISR5,
2802:     IOKern_ISR6,
2803:     IOKern_ISR7,
2804:     IOKern_ISR8,
2805:     IOKern_ISR9,
2806:     IOKern_ISR10,
2807:     IOKern_ISR11,
2808:     IOKern_ISR12,
2809:     IOKern_ISR13,
2810:     IOKern_ISR14,
2811:     IOKern_ISR15,
2812:     IOKern_ISR16,
2813:     NULL
2814: };
2815:
2816: int IOKern_IRQ_Test_Helper( unsigned int irq )
2817: {
2818:     int error_code = SUCCESS;
2819: #if defined( IOKERN_CPU_REGION_MAP )
2820:     int index;
2821:
2822:     index = 0;
2823:     while( SYS_TYPE__NONE != iokern_irq_map_list[index].type )
2824:     {
2825:         if ( irq == iokern_irq_map_list[index].irq )
2826:         {
2827:             return( -((int)(index + 1)) );
2828:         }
2829:         index++;
2830:     }
2831: #endif
2832:
2833: switch( ( IOKERN_IRQ_ENUM ) irq )
2834: {
2835:     case IOKERN_IRQ_NONE:
2836:
2837:     case IOKERN_IRQ_1:
2838:
2839:     case IOKERN_IRQ_3:
2840:     case IOKERN_IRQ_4:
2841:     case IOKERN_IRQ_5:
2842:     case IOKERN_IRQ_6:
2843:     case IOKERN_IRQ_7:
2844:     case IOKERN_IRQ_9:
2845:     case IOKERN_IRQ_10:
2846:     case IOKERN_IRQ_11:
2847:     case IOKERN_IRQ_12:
2848:
2849:     case IOKERN_IRQ_14:
2850:     case IOKERN_IRQ_15:
2851:
2852:         break;
2853:     default:
```

```
2859:     error_code = -EC_INTR_ERROR;
2860: }
2861:
2862: return error_code;
2863: }
2864:
2870: int IOKern_IRQ_Test( unsigned int irq )
2871: {
2872:     int error_code;
2873:
2874:     error_code = IOKern_IRQ_Test_Helper( irq );
2875:
2877:     if ( error_code ) return -EC_INTR_ERROR;
2878:
2879:     return SUCCESS;
2880: }
2881:
2887: void IOKern_IRQ_Invoke_ISR_Test( size_t irq )
2888: {
2889:     struct pt_regs r;
2890:
2891:     memset( &r, 0, sizeof( struct pt_regs ) );
2892:     ( * iokern_isr_table[irq] )( r );
2893: }
2894:
2900: IOKERN_TASK_ID_ENUM IOKern_Task_ID_Get( unsigned int irq )
2901: {
2902:     IOKERN_TASK_ID_ENUM task_id;
2903:
2904:     if ( IOKERN_IRQ_NONE == irq )
2905:     {
2906:         task_id = IOKERN_TASK_ID_NONE;
2907:     }
2908:     else if ( irq <= IOKERN_IRQ_15 )
2909:     {
2910:         task_id = ( IOKERN_TASK_ID_ENUM ) irq;
2911:     }
2912:     else
2913:     {
2914:         task_id = IOKERN_TASK_ID_NONE;
2915:     }
2916:     return task_id;
2917: }
2918:
2924: IOKERN_TASK_TYPE * IOKern_Task_Handle_Get( unsigned int irq )
2925: {
2926:     size_t task_id;
2927:
2928:     if ( SUCCESS != IOKern_IRQ_Test( irq ) ) return( NULL );
2929:
2930:     task_id = 0;
2931:     while( task_id < IOKERN_TASK_QTY )
2932:     {
2933:         if ( irq == iokern_task[task_id].number )
2934:         {
2935:             return( &iokern_task[task_id] );
2936:         }
2937:         task_id++;
2938:     }
2939:
2940:     return NULL;
2941: }
2942:
2948: static int IOKern_DOS_ISR_Install( unsigned char irq )
```

```
2949: {
2950:     IOKERN_TASK_ID_ENUM task_id;
2951:     int int_number;
2952:
2953:     task_id = IOKern_Task_ID_Get( irq );
2954:     if ( IOKERN_TASK_ID_NONE == task_id ) return SUCCESS;
2955:
2956:     if ( ( IOKERN_TASK_ID_IRQ0 == task_id ) || ( IOKERN_TASK_ID_INT33 == task_id ) )
2957:     {
2958:         return -EC_INTERRUPT_UNAVAILABLE;
2959:     }
2960:
2961:
2962:
2963:
2964:
2965:
2966:
2967:
2968:
2969:
2970:
2971:
2972:
2973:
2974:
2975:     if ( irq < 8 ) int_number = ( irq - 0 ) + 0x08;
2976:     else           int_number = ( irq - 8 ) + 0x70;
2977:
2978:
2979:     iokern_task[task_id].old_isr = (IOKERN_ISR_FP) IOKERN_DOS_INT_GET( int_number );
2980:     IOKern_DOS_IRQ_Mask_Get( irq, &(iokern_task[task_id].old_state) );
2981:     IOKERN_DOS_INT_SET( int_number, iokern_isr_table[task_id] );
2982:     IOKern_DOS_IRQ_Mask_Set( irq, 0 );
2983:
2984:
2985:     return SUCCESS;
2986: }
2987:
2993: static void IOKern_DOS_ISR_Restore( unsigned char irq )
2994: {
2995:     IOKERN_TASK_ID_ENUM task_id;
2996:     int int_number;
2997:
2998:     task_id = IOKern_Task_ID_Get( irq );
2999:     if ( IOKERN_TASK_ID_NONE == task_id ) return;
3000:
3001:
3002:
3003:
3004:
3005:
3006:
3007:
3008:
3009:     if ( irq < 8 ) int_number = ( irq - 0 ) + 0x08;
3010:     else           int_number = ( irq - 8 ) + 0x70;
3011:
3012:     IOKern_DOS_IRQ_Mask_Set( irq, iokern_task[task_id].old_state );
3013:     IOKERN_DOS_INT_SET( int_number, iokern_task[task_id].old_isr );
3014:
3015: }
3016:
3022: int IOKern_DOS_IRQ_Request( unsigned int irq,
```

```
3023:     IOKERN_TASK_FP handler,
3024:
3025:
3026:     void *    dev_id
3027:     )
3028: {
3029:     int      error_code;
3030:     IOKERN_TASK_ID_ENUM   task_id;
3031: #define IOKERN_FUNCTION_NAME "IOKern_DOS_IRQ_Request"
3032: #if defined ( MSDOS_PRINTF_DEBUG_MESSAGES )
3033:     Printf( "%s: ", IOKERN_FUNCTION_NAME );
3034:     Printf( "irq=%d, ", irq );
3035: #endif
3036: #undef IOKERN_FUNCTION_NAME
3037:     if ( SUCCESS != IOKern_IRQ_Test( irq ) ) return -EC_INTR_ERROR;
3038:
3039:     task_id = IOKern_Task_ID_Get( irq );
3040:     if ( IOKERN_TASK_ID_NONE == task_id ) return SUCCESS;
3041:
3042:     if ( NULL == handler ) return -EC_INTR_ERROR;
3043:     if ( NULL != iokern_task[task_id].task_fp ) return -EC_BUSY;
3044:
3045:     iokern_task[task_id].task_fp = handler;
3046:
3047:
3048:
3049:
3050:
3051:
3052:
3053:
3054:
3055:
3056:
3057:
3058:
3059:
3060:
3061:
3062:
3063:
3064:
3065:     iokern_task[task_id].help_fp  = NULL;
3066:     iokern_task[task_id].chain_to_old = IOKERN_CHAIN_TO_OLD_OFF;
3067:
3068:
3069:     iokern_task[task_id].dev_id  = dev_id;
3070:
3071:     iokern_task[task_id].number  = irq;
3072:     error_code = IOKern_DOS_ISR_Install( (unsigned char) irq );
3073: #if defined ( MSDOS_PRINTF_DEBUG_MESSAGES )
3074:     Printf( "done\n" );
3075: #endif
3076:     return error_code;
3077: #undef IOKERN_FUNCTION_NAME
3078: }
3079:
3080: void IOKern_DOS_IRQ_Free( unsigned int  irq
3081:
3082:     )
3083: {
3084:     IOKERN_TASK_TYPE *  task;
3085:
3086: #define IOKERN_FUNCTION_NAME "IOKern_DOS_IRQ_Free"
```

```

3092: #if defined ( MSDOS_PRINTF_DEBUG_MESSAGES )
3093: Printf( "%s: ", IOKERN_FUNCTION_NAME );
3094: Printf( "irq=%d, ", irq );
3095: #endif
3096: task = IOKern_Task_Handle_Get( irq );
3097: if ( ( SUCCESS == IOKern IRQ_Test( irq ) ) && ( NULL != task ) )
3098: {
3099:   if ( task->task_fp )
3100:   {
3101:     IOKern_DOS_ISR_Restore( (unsigned char) irq );
3102:     memset( task, 0, sizeof( IOKERN_TASK_TYPE ) );
3103:     task->number = IOKERN_IRQ_NONE;
3104: #if defined ( MSDOS_PRINTF_DEBUG_MESSAGES )
3105:   Printf( "success" );
3106: #endif
3107: }
3108: }
3109: else
3110: {
3111: #if defined ( MSDOS_PRINTF_DEBUG_MESSAGES )
3112:   Printf( " " );
3113: #endif
3114: }
3115: #if defined ( MSDOS_PRINTF_DEBUG_MESSAGES )
3116:   Printf( "done\n" );
3117: #endif
3118: #undef IOKERN_FUNCTION_NAME
3119: }
3120:
3122: void IOKern_Resource_Termination( void )
3123: {
3124:   size_t index;
3125:
3126:   IOKERN_IRQ_START;
3127:   for ( index = 0; index < IOKERN_TASK_QTY; index++ )
3128:   {
3129:     IOKern_DOS_IRQ_Free( iokern_task[index].number );
3130:   }
3131:   IOKERN_IRQ_ENABLE;
3132: }
3133:
3135: void IOKern_Resource_Initialization( void )
3136: {
3137:   size_t index;
3138:
3139:
3140:
3141:
3142:
3143:   memset( iokern_task, 0, IOKERN_TASK_QTY * sizeof( IOKERN_TASK_TYPE ) );
3144:
3145:   for ( index = 0; index < IOKERN_TASK_QTY; index++ )
3146:   {
3147:     iokern_task[index].number = IOKERN_IRQ_NONE;
3148:   }
3149:
3150: #if defined( IOKERN_CPU_REGION_MAP )
3151:
3152:   index = 0;
3153:   while( SYS_TYPE_NONE != iokern_cpu_region_list[index].type )
3154:   {
3155:     iokern_cpu_region_list[index].name = iokern_cpu_region_description[index];
3156:     index++;
3157:   }

```

```
3158:
3159:     index = 0;
3160:     while( SYS_TYPE__NONE != iokern_irq_map_list[index].type )
3161:     {
3162:         iokern_irq_map_list[index].name = iokern_irq_map_description[index];
3163:         index++;
3164:     }
3165: #endif
3166: }
3167:
3168:
3169: #endif
3170:
3171:
3172:
3173: #if(0)
3174:
3175:
3176:
3177:
3178:
3179:
3180:
3189: int IAI_AI_ID_Get( int board_id, uint16_t * id )
3190: {
3191:     uint8_t     lsb, msb;
3192:
3193:     IO_Read_U8( board_id, IAI_ID_LSB, &lsb );
3194:     IO_Read_U8( board_id, IAI_ID_MSB, &msb );
3195:     *id = ( ((uint16_t) msb) << 8 ) | ((uint16_t) lsb);
3196: #if defined( ID_ALWAYS_REPORT_AS_GOOD )
3197:     *id = ID_IAI;
3198: #endif
3199:     return SUCCESS;
3200: }
3201:
3206: int IAI16_AI_Channel_Get( int board_id, size_t channel, int * value )
3207: {
3208:     int     location;
3209:     size_t   row;
3210:     uint16_t scratch;
3211:     struct reg_definition * definition = ( struct reg_definition * )
board_definition[board_id].definition;
3212:
3213:     row = REG_LOCATION_LOGICAL_GET( IAI_AIN0 ) + channel;
3214:     location = REG_LOCATION_SET( row, 0 );
3215:
3216:     IO_Read_U16_Address_Increment( board_id, location, &scratch );
3217:
3218:     *value = (int) scratch;
3219:     return SUCCESS;
3220: }
3221:
3226: int IAI16_AI_Channel_Multiple_Get( int board_id,
3227:                                     size_t   channel_first,
3228:                                     size_t   channel_last,
3229:                                     size_t   table_offset,
3230:                                     int *    table
3231:                                     )
3232: {
3233:     int     location;
3234:     size_t   row;
3235:     size_t   index;
3236:     uint16_t scratch;
```

```
3237: struct reg_definition * definition = ( struct reg_definition * )
board_definition[board_id].definition;
3238:
3239: index = 0;
3240: for ( channel = channel_first; channel <= channel_last; channel++ )
3241: {
3242:   row = REG_LOCATION_LOGICAL_GET( IAI_AIN0 ) + channel;
3243:   location = REG_LOCATION_SET( row, 0 );
3244:   IO_Read_U16_Address_Increment( board_id, location, &scratch );
3245:
3246:   value[table_offset + index] = (int) scratch;
3247:   index++;
3248: }
3249: return SUCCESS;
3250: }
3251:
3252:
3253:
3254:
3255:
3260:
3261:
3262:
3263:
3268:
3269:
3270:
3275:
3276:
3277:
3283: static int CMD__IAI16_AI_ID( int board_id, int argc, char * argv[] )
3284: {
3285:   uint16_t id;
3286:   (void) argc;
3287:   (void) argv;
3288:   IAI_AI_ID_Get( board_id, &id );
3289:   printf( "IAI16 AI ID: 0x%04X\n", id );
3290:   return SUCCESS;
3291: }
3292:
3298: static int CMD__IAI16_AI_Channel( int board_id, int argc, char * argv[] )
3299: {
3300:   int error_code;
3301:   int channel;
3302:   int value;
3303:
3304:   if ( argc < 1 ) return -EC_NOT_FOUND;
3305:
3306:   channel = (int) strtol( argv[0], NULL, 0 );
3307:
3308:   if ( argc < 2 )
3309:   {
3310:     error_code = IAI_AI_Channel_Get( board_id, channel, &value );
3311:     printf( "IAI%02d: 0x%04X %d\n", channel, value, value );
3312:   }
3313:   return SUCCESS;
3314: }
3315:
3321: static int CMD__IAI16_AI_All( int board_id, int argc, char * argv[] )
3322: {
3323:   int error_code;
3324:   int channel;
3325:   int value;
3326:
```

```
3327: if ( argc < 1 ) return -EC_NOT_FOUND;
3328:
3329: channel = (int) strtol( argv[0], NULL, 0 );
3330:
3331:
3332: IAI16_AI_Channel_Multiple_Get( board_id,
3333:                                 0,
3334:                                 IAI16_CHANNEL_QTY - 1,
3335:                                 0,
3336:                                 table
3337:                               );
3338:
3339: for ( channel = 0; channel < IAI16_CHANNEL_QTY; channel++ )
3340: {
3341:   printf( "IA%02d: 0x%04X %d\n", channel, table[channel], table[channel] );
3342: }
3343:
3344:
3345:
3346:
3347:
3348:
3349:
3350:
3351: return SUCCESS;
3352: }
3353:
3354:
3355:
3359: static struct command_line_board cmd_iai16[] =
3360: {
3361:   { NULL,     CMD__IAI16_AI_ID,  "id",    "params: none. Reports the AI board/component
ID." },
3362:   { NULL,     CMD__IAI16_AI_Channel, "chan",  "params: <channel>" },
3363:   { NULL,     CMD__IAI16_AI_All,   "all",   "gets all analog inputs" },
3364:   { cmd_ido48_test, CMD__IAI16_Test, "test",  "semi-auto test functions" },
3365:   { NULL, NULL, NULL, NULL },
3366: };
3367:
3373: int Command_Line_IAI16( int board_id, int argc, char* argv[] )
3374: {
3375:   int error_code;
3376:   int index;
3377:   int argc_new;
3378:   char ** argv_new;
3379:   char * endptr;
3380:
3381:
3382:   if ( argc < 1 ) return -EC_NOT_FOUND;
3383:
3384:   if ( ID_IDO48 != board_id ) return -EC_BOARD_TYPE;
3385:
3386:   error_code = -EC_SYNTAX;
3387:
3388:
3389:   strtol( argv[0], &endptr, 0 );
3390:   if ( argv[0] != endptr )
3391:   {
3392:     error_code = (* cmd_iai16[0].cmd_fnc )( board_id, argc, argv );
3393:   }
3394:   else
3395:   {
3396:     index = 0;
3397:     while ( NULL != cmd_iai16[index].cmd_fnc )
```

```
3398: {
3399:     if ( 0 == strcmpi( cmd_iai16[index].name, argv[0] ) )
3400:     {
3401:         argv_new = &(argv[1]);
3402:         argc_new = argc - 1;
3403:         if ( 0 == argc_new ) argv_new = NULL;
3404:         error_code = (* cmd_iai16[index].cmd_fnc )( board_id, argc_new, argv_new );
3405:         break;
3406:     }
3407:     index++;
3408: }
3409: }
3410: return error_code;
3411: }
3412:
3413:
3414:
3415:
3416:
3417:
3418:
3419: #endif
3420:
3421:
3422:
3423:
3424:
3425:
3426:
3427:
3436: int IDI_DIN_ID_Get( int board_id, uint16_t * id )
3437: {
3438:     uint8_t lsb, msb;
3439:     struct board_dataset * dataset;
3440:
3441:     dataset = (struct board_dataset *) board_definition[board_id].dataset;
3442:
3443:     if ( MODE_JUMPERS_0 == dataset->mode_jumpers )
3444:     {
3445:         return -EC_MODE_LEGACY;
3446:     }
3447:
3448:     IO_Read_U8( board_id, __LINE__, IDI_ID_LSB, &lsb );
3449:     IO_Read_U8( board_id, __LINE__, IDI_ID_MSB, &msb );
3450:     *id = ( ((uint16_t) msb) << 8 ) | ((uint16_t) lsb);
3451:
3452:
3453: #if defined( ID_ALWAYS_REPORT_AS_GOOD )
3454:     *id = ID_DIN;
3455: #endif
3456:     return SUCCESS;
3457: }
3458:
3466: BOOL IDI_DIN_IsNotPresent( int board_id )
3467: {
3468:     uint16_t id;
3469:     IDI_DIN_ID_Get( board_id, &id );
3470:     if (ID_DIN == id ) return 0;
3471:     return 1;
3472: }
3473:
3483: int IDI_DIN_Channel_Get( int board_id, size_t channel, BOOL * value )
3484: {
3485:     size_t group;
```

```
3486: size_t bit;
3487: uint8_t reg_value;
3488:
3489: group = channel >> IDI_DIN_SHIFT_RIGHT;
3490: bit = channel - group * IDI_DIN_GROUP_SIZE;
3491:
3492: IO_Read_U8( board_id, __LINE__, IDI_DI_GROUP0 + group , &reg_value );
3493:
3494: if ( 0 != ( reg_value & ( 0x01 << bit ) ) ) *value = true;
3495: else *value = false;
3496:
3497: return SUCCESS;
3498: }
3499:
3509: int IDI_DIN_Group_Get( int board_id, size_t group, uint8_t * value )
3510: {
3511: IO_Read_U8( board_id, __LINE__, IDI_DI_GROUP0 + group , value );
3512: return SUCCESS;
3513: }
3514:
3523: int IDI_DIN_Pending_Clear( int board_id, size_t channel )
3524: {
3525: size_t group;
3526: size_t bit;
3527: uint8_t reg_value;
3528:
3529: group = channel >> IDI_DIN_SHIFT_RIGHT;
3530: bit = channel - group * IDI_DIN_GROUP_SIZE;
3531:
3532: if ( group >= IDI_DIN_GROUP_QTY ) return -EC_CHANNEL;
3533:
3534: reg_value = (uint8_t)( 0x01 << bit );
3535: IO_Write_U8( board_id, __LINE__, IDI_CLEAR_GROUP0 + group , reg_value );
3536: return SUCCESS;
3537: }
3538:
3539:
3540:
3541:
3542:
3543:
3544:
3546:
3548:
3549:
3558: static int SPI_ID_Get_Helper( int board_id, int id_lsb, int id_msb, uint16_t * id )
3559: {
3560: uint8_t lsb, msb;
3561: struct board_dataset * dataset;
3562:
3563: dataset = (struct board_dataset *) board_definition[board_id].dataset;
3564:
3565: #define FUNCTION_NAME__ "SPI_ID_Get_Helper"
3566: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
3567: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
3568: #endif
3569: #undef FUNCTION_NAME__
3570:
3571: if ( MODE_JUMPERS_0 == dataset->mode_jumpers )
3572: {
3573: return -EC_MODE_LEGACY;
3574: }
3575:
3576: IO_Read_U8( board_id, __LINE__, id_lsb, &lsb );
```

```

3577: IO_Read_U8( board_id, __LINE__, id_msb, &msb );
3578: *id = ( ((uint16_t) msb) << 8 ) | ((uint16_t) lsb);
3579: #if defined( ID_ALWAYS_REPORT_AS_GOOD )
3580: *id = ID_SPI;
3581: #endif
3582: return SUCCESS;
3583: }
3584:
3585: int SPI_ID_Get( int board_id, uint16_t * id )
3586: {
3587: #define FUNCTION_NAME__ "SPI_ID_Get"
3588: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
3589: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
3590: #endif
3591: #undef FUNCTION_NAME__
3592:
3593: switch ( board_id )
3594: {
3595: case ID_IDI48:
3596:     return SPI_ID_Get_Helper( board_id, IDI_SPI_ID_LSB, IDI_SPI_ID_MSB, id );
3597: case ID_IDO48:
3598:     return SPI_ID_Get_Helper( board_id, IDO_SPI_ID_LSB, IDO_SPI_ID_MSB, id );
3599: }
3600: return -EC_BOARD;
3601: }
3602:
3611: int SPI_IsNotPresent( int board_id )
3612: {
3613:     int         error_code;
3614:     uint16_t    id;
3615:     struct board_dataset * dataset;
3616:     dataset = (struct board_dataset *) board_definition[board_id].dataset;
3617:
3618: #define FUNCTION_NAME__ "SPI_IsNotPresent"
3619: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
3620: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
3621: #endif
3622: #undef FUNCTION_NAME__
3623:
3624:
3627: if ( true == dataset->spi_is_present ) return 0;
3628:
3629: error_code = SPI_ID_Get( board_id, &id );
3630: if ( error_code < 0 ) return error_code;
3631: if ( ID_SPI == id )
3632: {
3633:     dataset->spi_is_present = true;
3634:     return 0;
3635: }
3636: return 1;
3637: }
3638:
3653: int SPI_Calculate_Half_Clock( double      half_clock_request_sec,
3654:                                double *    half_clock_actual_sec,
3655:                                double *    error,
3656:                                uint16_t *  hci
3657:                                )
3658: {
3659:     double      scratch;
3660:     int        hci_temp;
3661:
3662:
3663:     scratch = ( half_clock_request_sec / CLOCK_PERIOD_SEC ) - 4.0;
3664:     hci_temp = (int) scratch;

```

```
3665:
3666: if ( ( hci_temp > 4095 ) || ( hci_temp < 0 ) ) return -EC_SPI_HALF_CLOCK_OUT_OF_RANGE;
3667:
3668:
3669: scratch = CLOCK_PERIOD_SEC * ( 4.0 + ((double) hci_temp) );
3670: if ( NULL != error ) *error = ( scratch -
half_clock_request_sec ) / half_clock_request_sec;
3671: if ( NULL != half_clock_actual_sec ) *half_clock_actual_sec = scratch;
3672: if ( NULL != hci ) *hci = (uint16_t) hci_temp;
3673: return SUCCESS;
3674: }
3675:
3690: int SPI_Calculate_Clock( double clock_request_hz,
3691:                           double *clock_actual_hz,
3692:                           double *error,
3693:                           uint16_t *hci
3694:                         )
3695: {
3696:     int error_code;
3697:     double half_clock_request_sec;
3698:     double half_clock_actual_sec;
3699:     double error_internal;
3700:     double scratch;
3701:     uint16_t hci_internal;
3702:
3703:     half_clock_request_sec = 1.0 / ( 2.0 * clock_request_hz );
3704:
3705:     error_code = SPI_Calculate_Half_Clock( half_clock_request_sec,
3706:                                           &half_clock_actual_sec,
3707:                                           &error_internal,
3708:                                           &hci_internal
3709:                                         );
3710:     if ( SUCCESS != error_code ) return error_code;
3711:
3712:
3713:     scratch = 1.0 / ( 2.0 * half_clock_actual_sec );
3714:     if ( NULL != error ) *error = ( scratch - clock_request_hz ) /
clock_request_hz;
3715:     if ( NULL != clock_actual_hz ) *clock_actual_hz = scratch;
3716:     if ( NULL != hci ) *hci = (uint16_t) hci_internal;
3717:     return SUCCESS;
3718: }
3719:
3727: double SPI_Calculate_Half_Clock_Interval_Sec( uint16_t half_clock_interval )
3728: {
3729:     double half_clock_interval_sec;
3730:     half_clock_interval_sec = CLOCK_PERIOD_SEC * ( 4.0 + ((double) half_clock_interval) );
3731:     return half_clock_interval_sec;
3732: }
3733:
3747: int SPI_Calculate_End_Cycle_Delay( double spi_half_clock_interval_sec,
3748:                                     double delay_request_sec,
3749:                                     double *delay_actual_sec,
3750:                                     double *error,
3751:                                     uint8_t *ecd
3752:                                   )
3753: {
3754:
3755:     double scratch;
3756:     int ecd_temp;
3757:
3758:
3759:     scratch = ( delay_request_sec - 4.0 * CLOCK_PERIOD_SEC ) / spi_half_clock_interval_sec;
3760:     ecd_temp = (int) scratch;
```

```

3761:
3762:     if ( (ecd_temp > 255) || (ecd_temp < 0) ) return -EC_SPI_ECD_OUT_OF_RANGE;
3763:
3764:
3765:     scratch = CLOCK_PERIOD_SEC * 4.0 + ((double)ecd_temp) * spi_half_clock_interval_sec;
3766:     if (NULL != error) *error = (scratch - delay_request_sec) / delay_request_sec;
3767:     if (NULL != delay_actual_sec) *delay_actual_sec = scratch;
3768:     if (NULL != ecd) *ecd = (uint8_t)ecd_temp;
3769:     return SUCCESS;
3770: }
3771:
3780: int SPI_Configuration_Chip_Select_Behavior_Get( int board_id, SPI_CSBOARD_ENUM *chip_select_behavior )
3781: {
3782:     uint8_t scratch;
3783:
3784: #define FUNCTION_NAME__ "SPI_Configuration_Chip_Select_Behavior_Get"
3785: #if (IO_TRACE_USE_SPI_LOGGING == 1)
3786:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
3787: #endif
3788: #undef FUNCTION_NAME__
3789:
3790:     if (SPI_IsNotPresent(board_id)) return -EC_SPI_NOT_FOUND;
3791:
3792:     switch (board_id)
3793:     {
3794:         case ID_IDI48: IO_Read_U8(board_id, __LINE__, IDI_SPI_CONFIG, &scratch); break;
3795:         case ID_IDO48: IO_Read_U8(board_id, __LINE__, IDO_SPI_CONFIG, &scratch); break;
3796:     }
3797:
3798:     *chip_select_behavior = (SPI_CSBOARD_ENUM)(scratch >> 4) & 0x03;
3799:     return SUCCESS;
3800: }
3801:
3810: int SPI_Configuration_Chip_Select_Behavior_Set( int board_id, SPI_CSBOARD_ENUM chip_select_behavior )
3811: {
3812:     uint8_t scratch;
3813:     int spi_config_symbol;
3814:
3815: #define FUNCTION_NAME__ "SPI_Configuration_Chip_Select_Behavior_Set"
3816: #if (IO_TRACE_USE_SPI_LOGGING == 1)
3817:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
3818: #endif
3819: #undef FUNCTION_NAME__
3820:
3821:     if (SPI_IsNotPresent(board_id)) return -EC_SPI_NOT_FOUND;
3822:
3823:     switch (board_id)
3824:     {
3825:         case ID_IDI48: spi_config_symbol = (int)IDI_SPI_CONFIG; break;
3826:         case ID_IDO48: spi_config_symbol = (int)IDO_SPI_CONFIG; break;
3827:     }
3828:
3829:     IO_Read_U8(board_id, __LINE__, spi_config_symbol, &scratch);
3830:
3831:     scratch &= ~0x30;
3832:     scratch |= (uint8_t)((chip_select_behavior & 0x03) << 4);
3833:
3834:     IO_Write_U8(board_id, __LINE__, spi_config_symbol, scratch);
3835:     return SUCCESS;
3836: }
3837:
```

```
3846: int SPI_Configuration_Chip_Select_Route_Set( int board_id, BOOL cs_to_channel1 )
3847: {
3848:     int spi_config_symbol;
3849:     uint8_t scratch;
3850:     uint8_t mask = 0x40;
3851:
3852: #define FUNCTION_NAME__ "SPI_Configuration_Chip_Select_Route_Set"
3853: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
3854:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
3855: #endif
3856: #undef FUNCTION_NAME__
3857:
3858: if ( SPI_IsNotPresent( board_id ) ) return -EC_SPI_NOT_FOUND;
3859:
3860: switch ( board_id )
3861: {
3862:     case ID_IDI48: spi_config_symbol = (int) IDI_SPI_CONFIG; break;
3863:     case ID_IDO48: spi_config_symbol = (int) IDO_SPI_CONFIG; break;
3864: }
3865:
3866: IO_Read_U8( board_id, __LINE__, spi_config_symbol, &scratch );
3867:
3868: if ( true == cs_to_channel1 ) scratch |= mask;
3869: else
3870:     scratch &= ~mask;
3871:
3872: IO_Write_U8( board_id, __LINE__, spi_config_symbol, scratch );
3873: return SUCCESS;
3874:
3883: int SPI_Configuration_Chip_Select_Route_Get( int board_id, BOOL * cs_to_channel1 )
3884: {
3885:     uint8_t scratch;
3886:     uint8_t mask = 0x40;
3887:
3888: #define FUNCTION_NAME__ "SPI_Configuration_Chip_Select_Route_Get"
3889: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
3890:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
3891: #endif
3892: #undef FUNCTION_NAME__
3893:
3894: if ( SPI_IsNotPresent( board_id ) ) return -EC_SPI_NOT_FOUND;
3895:
3896: switch ( board_id )
3897: {
3898:     case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_CONFIG, &scratch ); break;
3899:     case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_CONFIG, &scratch ); break;
3900: }
3901:
3902: if ( scratch & mask ) *cs_to_channel1 = true;
3903: else
3904:     *cs_to_channel1 = false;
3905:
3906: return SUCCESS;
3907:
3916: int SPI_Configuration_Set( int board_id, struct spi_cfg * cfg )
3917: {
3918:     int error_code;
3919:     double scratch;
3920:     double hci_sec;
3921:     uint8_t config;
3922:     struct board_dataset * dataset = ( struct board_dataset * )
board_definition[board_id].dataset;
3923:
3924: #define FUNCTION_NAME__ "SPI_Configuration_Set"
```

```

3925: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
3926: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
3927: #endif
3928: #undef FUNCTION_NAME__
3929:
3930: if ( SPI_IsNotPresent( board_id ) ) return -EC_SPI_NOT_FOUND;
3931:
3932: config = (uint8_t) ( (cfg->chip_select_behavior & 0x03) << 4 );
3933: if ( cfg->sclk_polarity ) config |= 0x01;
3934: if ( cfg->sclk_phase ) config |= 0x02;
3935: if ( cfg->sdi_polarity ) config |= 0x04;
3936: if ( cfg->sdo_polarity ) config |= 0x08;
3937: if ( cfg->sdio_wrap ) config |= 0x80;
3938: if ( cfg->chip_select_route_ch1 ) config |= 0x40;
3939:
3940: switch ( board_id )
3941: {
3942: case ID_IDI48: IO_Write_U8( board_id, __LINE__, IDI_SPI_CONFIG, config ); break;
3943: case ID_IDO48: IO_Write_U8( board_id, __LINE__, IDO_SPI_CONFIG, config ); break;
3944: }
3945:
3946:
3947: if ( cfg->clock_hz > 0 )
3948: {
3949:
3950: error_code = SPI_Calculate_Clock( cfg->clock_hz, NULL, NULL,
&(cfg->half_clock_interval) );
3951: if ( SUCCESS != error_code ) return error_code;
3952: }
3953: hci_sec = SPI_Calculate_Half_Clock_Interval_Sec( cfg->half_clock_interval );
3954:
3955: if ( cfg->end_delay_ns > 0 )
3956: {
3957: scratch = cfg->end_delay_ns * 1.0e-9;
3958: error_code = SPI_Calculate_End_Cycle_Delay( hci_sec,
3959:                                             scratch,
3960:                                             NULL,
3961:                                             NULL,
3962:                                             &(cfg->end_cycle_delay)
3963: );
3964: if ( SUCCESS != error_code ) return error_code;
3965: }
3966:
3967: switch ( board_id )
3968: {
3969: case ID_IDI48: IO_Write_U8( board_id, __LINE__, IDI_SPI_HCI_LSB, (uint8_t)(
cfg->half_clock_interval & 0xFF ) ); break;
3970: case ID_IDO48: IO_Write_U8( board_id, __LINE__, IDO_SPI_HCI_LSB, (uint8_t)(
cfg->half_clock_interval & 0xFF ) ); break;
3971: }
3972:
3973:
3974:
3975: switch ( board_id )
3976: {
3977: case ID_IDI48: IO_Write_U8( board_id, __LINE__, IDI_SPI_HCI_MSB, (uint8_t)(
cfg->half_clock_interval >> 8 ) ); break;
3978: case ID_IDO48: IO_Write_U8( board_id, __LINE__, IDO_SPI_HCI_MSB, (uint8_t)(
cfg->half_clock_interval >> 8 ) ); break;
3979: }
3980:
3981:
3982: switch ( board_id )
3983: {

```

```
3984: case ID_IDI48: IO_Write_U8( board_id, __LINE__, IDI_SPI_ECD, cfg->end_cycle_delay );
break;
3985: case ID_IDO48: IO_Write_U8( board_id, __LINE__, IDO_SPI_ECD, cfg->end_cycle_delay );
break;
3986: }
3987:
3988:
3989: if ( cfg != &(dataset->spi.cfg) )
3990: {
3991:     memcpy( &(dataset->spi.cfg), &cfg, sizeof( struct spi_cfg ) );
3992: }
3993: return SUCCESS;
3994: }
3995:
4004: int SPI_Configuration_Get( int board_id, struct spi_cfg * cfg )
4005: {
4006:     uint8_t scratch;
4007:     struct board_dataset * dataset = ( struct board_dataset * )
board_definition[board_id].dataset;
4008:
4009: #define FUNCTION_NAME__ "SPI_Configuration_Get"
4010: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
4011:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
4012: #endif
4013: #undef FUNCTION_NAME__
4014:
4015: if ( SPI_IsNotPresent( board_id ) ) return -EC_SPI_NOT_FOUND;
4016:
4017: switch ( board_id )
4018: {
4019:     case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_CONFIG, &scratch ); break;
4020:     case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_CONFIG, &scratch ); break;
4021: }
4022:
4023:
4024: cfg->chip_select_route_ch1 = ( scratch & 0x40 ) ? true : false;
4025: cfg->chip_select_behavior = (SPI_CSB_ENUM) ( scratch >> 4 ) & 0x03;
4026:     cfg->sclk_polarity = ( scratch & 0x01 ) ? true : false;
4027:     cfg->sclk_phase = ( scratch & 0x02 ) ? true : false;
4028:     cfg->sdi_polarity = ( scratch & 0x04 ) ? true : false;
4029:     cfg->sdo_polarity = ( scratch & 0x08 ) ? true : false;
4030:     cfg->sdio_wrap = ( scratch & 0x80 ) ? true : false;
4031:
4032: switch ( board_id )
4033: {
4034:     case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_HCI_LSB, &scratch ); break;
4035:     case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_HCI_LSB, &scratch ); break;
4036: }
4037:
4038: cfg->half_clock_interval = (uint16_t) scratch;
4039: switch ( board_id )
4040: {
4041:     case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_HCI_MSB, &scratch ); break;
4042:     case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_HCI_MSB, &scratch ); break;
4043: }
4044:
4045: cfg->half_clock_interval |= ( (uint16_t) scratch) << 8;
4046: switch ( board_id )
4047: {
4048:     case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_ECD, &(cfg->end_cycle_delay) );
break;
4049:     case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_ECD, &(cfg->end_cycle_delay) );
break;
4050: }
```

```

4051:
4052:
4053: cfg->clock_hz = 1.0 / ( 2.0 * CLOCK_PERIOD_SEC * ( 4.0 + ((double)
cfg->half_clock_interval) ) );
4054: cfg->end_delay_ns = 1.0e9 * CLOCK_PERIOD_SEC * 4.0 + 0.5 * ((double)
cfg->end_cycle_delay) / cfg->clock_hz;
4055:
4056: if ( cfg != &(dataset->spi.cfg) )
4057: {
4058:     memcpy( &(dataset->spi.cfg), &cfg, sizeof( struct spi_cfg ) );
4059: }
4060: return SUCCESS;
4061: }
4062:
4072: int SPI_Report_Configuration_Text( struct spi_cfg * cfg, FILE * out )
4073: {
4074:     fprintf( out, "##### SPI Configuration:\n" );
4075:     fprintf( out, "sdio_wrap          = %s\n", cfg->sdio_wrap ? "true" : "false"
);
4076:     fprintf( out, "sdo_polarity        = %s\n", cfg->sdo_polarity ? "true" : "false"
);
4077:     fprintf( out, "sdi_polarity        = %s\n", cfg->sdi_polarity ? "true" : "false"
);
4078:     fprintf( out, "sclk_phase          = %s\n", cfg->sclk_phase ? "true" : "false" );
4079:     fprintf( out, "sclk_polarity       = %s\n", cfg->sclk_polarity ? "true" : "false"
);
4080:     fprintf( out, "chip_select_route_ch1 = %s\n", cfg->chip_select_route_ch1 ? "true" :
"false" );
4081:     fprintf( out, "chip_select_behavior = " );
4082:
4083:     switch( cfg->chip_select_behavior )
4084:     {
4085:         case CSB_SOFTWARE: fprintf( out, "CSB_SOFTWARE" ); break;
4086:         case CSB_BUFFER:   fprintf( out, "CSB_BUFFER" ); break;
4087:         case CSB_uint8_t:  fprintf( out, "CSB_uint8_t" ); break;
4088:         case CSB_uint16_t: fprintf( out, "CSB_uint16_t" ); break;
4089:         default:           fprintf( out, "undefined" ); break;
4090:     }
4091:     fprintf( out, "\n" );
4092:
4093:     fprintf( out, "end_cycle_delay      = 0x%02X (%d)\n", cfg->end_cycle_delay,
cfg->end_cycle_delay );
4094:     fprintf( out, "half_clock_interval   = 0x%04X (%d)\n", cfg->half_clock_interval,
cfg->half_clock_interval );
4095:
4096:     fprintf( out, "clock_hz              = %f Hz\n", cfg->clock_hz );
4097:     fprintf( out, "end_delay_ns          = %f ns\n", cfg->end_delay_ns );
4098:     fprintf( out, "\n" );
4099:     return SUCCESS;
4100: }
4101:
4111: int SPI_Report_Status_Text( struct spi_status * status, FILE * out )
4112: {
4113:     fprintf( out, "##### SPI " );
4114:     if ( status->tx_status ) fprintf( out, "TX" );
4115:     else                      fprintf( out, "RX" );
4116:
4117:     fprintf( out, " Status:\n" );
4118:
4119:     fprintf( out, "full      = %s\n", status->full ? "true" : "false" );
4120:     fprintf( out, "empty     = %s\n", status->empty ? "true" : "false" );
4121:     fprintf( out, "fifo size = %d\n", status->fifo_size );
4122:     fprintf( out, "fifo count= %d\n", status->fifo_count );
4123:

```

```
4124:     return SUCCESS;
4125: }
4126:
4135: int SPI_Configuration_Initialize( struct spi_cfg * cfg )
4136: {
4137:     cfg->sdio_wrap      = false;
4138:     cfg->sdo_polarity    = false;
4139:     cfg->sdi_polarity    = false;
4140:
4146:     cfg->scclk_phase    = false;
4147:     cfg->scclk_polarity  = false;
4148:     cfg->chip_select_route_ch1 = false;
4149:     cfg->chip_select_behavior = CSB_SOFTWARE;
4150:     cfg->end_cycle_delay  = 0;
4151:     cfg->half_clock_interval = 0;
4152:
4153:     cfg->clock_hz        = 0.0;
4154:     cfg->end_delay_ns     = 0.0;
4155:
4156:     return SUCCESS;
4157: }
4158:
4159:
4177: int SPI_Status_Write( int board_id, struct spi_status * status )
4178: {
4179:     uint8_t reg_tx_status;
4180:
4181: #define FUNCTION_NAME__ "SPI_Status_Write"
4182: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
4183:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
4184: #endif
4185: #undef FUNCTION_NAME__
4186:
4187:     if ( SPI_IsNotPresent( board_id ) ) return -EC_SPI_NOT_FOUND;
4188:
4189:     switch ( board_id )
4190:     {
4191:         case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_TX_STATUS, &reg_tx_status );
break;
4192:         case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_TX_STATUS, &reg_tx_status );
break;
4193:     }
4194:
4195:     status->fifo_count = (int) ( reg_tx_status & SPI_FIFO_SIZE_BIT_WIDTH );
4196:     status->full      = (BOOL) ( reg_tx_status & 0x80 );
4197:     status->fifo_size  = (int) SPI_FIFO_SIZE;
4198:     status->empty      = (BOOL) ( reg_tx_status & 0x40 );
4199:     status->tx_status   = true;
4200:     return SUCCESS;
4201: }
4202:
4219: void SPI_Status_Write_FIFO_Status( int board_id, BOOL * full, BOOL * empty, size_t * bytes_available )
4220: {
4221: #if defined( SPI_PRINT_XFR_DEBUG )
4222:     uint8_t reg_buf[4];
4223:     size_t scratch;
4224:     size_t index;
4225:     int error_code;
4226: #endif
4227:     uint8_t reg_tx_status;
4228:
4229: #define FUNCTION_NAME__ "SPI_Status_Write_FIFO_Status"
4230: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
```

```

4231: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
4232: #endif
4233: #undef FUNCTION_NAME__
4234:
4235:
4236: #if defined( SPI_PRINT_XFR_DEBUG )
4237: switch ( board_id )
4238: {
4239:   case ID_IDI48:
4240:     for ( index = 0; index < 4; index++ )
4241:       IO_Read_U8( board_id, __LINE__, IDI_SPI_TX_STATUS, &(reg_buf[index]) );
4242:     break;
4243:   case ID_IDO48:
4244:     for ( index = 0; index < 4; index++ )
4245:       IO_Read_U8( board_id, __LINE__, IDO_SPI_TX_STATUS, &(reg_buf[index]) );
4246:     break;
4247: }
4248:
4249: error_code = 0;
4250: for ( index = 1; index < 4; index++ )
4251: {
4252:   scratch = (size_t)( reg_buf[index] & SPI_FIFO_SIZE_BIT_WIDTH );
4253:   if ( scratch > ( (size_t)( reg_buf[0] & SPI_FIFO_SIZE_BIT_WIDTH ) ) )
4254:   {
4255:     error_code = -1;
4256:   }
4257:
4258:   scratch = (size_t) ( reg_buf[index] & 0x40 );
4259:   if ( scratch != ( (size_t) ( reg_buf[0] & 0x40 ) ) )
4260:   {
4261:     error_code = -1;
4262:   }
4263: }
4264:
4265: if ( error_code < 0 )
4266: {
4267:   printf( "###ERROR: SPI_Status_Write_FIFO_Status: " );
4268:   for ( index = 0; index < 4; index++ ) printf( ", 0x%02X", reg_buf[index] );
4269:   printf( "\n" );
4270: }
4271:
4272: reg_tx_status = reg_buf[0];
4273: #else
4274: switch ( board_id )
4275: {
4276:   case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_TX_STATUS, &reg_tx_status );
break;
4277:   case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_TX_STATUS, &reg_tx_status );
break;
4278: }
4279: #endif
4280:
4281: switch( reg_tx_status & 0xC0 )
4282: {
4283:   case 0x00: *full = false; *empty = false; break;
4284:   case 0x40: *full = false; *empty = true; break;
4285:   case 0x80: *full = true; *empty = false; break;
4286:   case 0xC0:
4287:     *full = true;
4288:     *empty = true;
4289: #define FUNCTION_NAME__ "SPI_Status_Write_FIFO_Status-FullEmpty"
4290: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
4291: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
4292: #endif

```

```
4293: #undef FUNCTION_NAME__
4294:
4295:     break;
4296: }
4297:
4298: *bytes_available = (size_t) SPI_FIFO_SIZE_BIT_WIDTH - (size_t)( reg_tx_status &
SPI_FIFO_SIZE_BIT_WIDTH );
4299: }
4300:
4308: BOOL SPI_Status_Write_FIFO_Is_Full( int board_id )
4309: {
4310:     uint8_t reg_tx_status;
4311:
4312: #define FUNCTION_NAME__ "SPI_Status_Write_FIFO_Is_Full"
4313: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
4314:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
4315: #endif
4316: #undef FUNCTION_NAME__
4317:
4318: switch ( board_id )
4319: {
4320:     case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_TX_STATUS, &reg_tx_status );
break;
4321:     case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_TX_STATUS, &reg_tx_status );
break;
4322: }
4323: if ( reg_tx_status & 0x80 ) return true;
4324: return false;
4325: }
4326:
4342: void SPI_Status_Read_FIFO_Status( int board_id, BOOL * empty, size_t * bytes_available
)
4343: {
4344: #if defined( SPI_PRINT_XFR_DEBUG )
4345:     uint8_t reg_value[4];
4346:     size_t index;
4347:     size_t scratch;
4348:     int     error_code;
4349: #endif
4350:     uint8_t reg_rx_status;
4351:
4352: #define FUNCTION_NAME__ "SPI_Status_Read_FIFO_Status"
4353: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
4354:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
4355: #endif
4356: #undef FUNCTION_NAME__
4357:
4358:
4359: #if defined( SPI_PRINT_XFR_DEBUG )
4360:     switch ( board_id )
4361:     {
4362:         case ID_IDI48:
4363:             for ( index = 0; index < 4; index++ )
4364:                 IO_Read_U8( board_id, __LINE__, IDI_SPI_RX_STATUS, &(reg_value[index]) );
4365:             break;
4366:         case ID_IDO48:
4367:             for ( index = 0; index < 4; index++ )
4368:                 IO_Read_U8( board_id, __LINE__, IDO_SPI_RX_STATUS, &(reg_value[index]) );
4369:             break;
4370:     }
4371:
4372:     error_code = 0;
4373:     for ( index = 1; index < 4; index++ )
4374:     {
```

```

4375:     scratch = (size_t)( reg_value[index] & SPI_FIFO_SIZE_BIT_WIDTH );
4376:     if ( scratch < ( (size_t)( reg_value[0] & SPI_FIFO_SIZE_BIT_WIDTH ) ) )
4377:     {
4378:         error_code = -1;
4379:     }
4380:
4381:     scratch = (size_t) ( reg_value[index] & 0x40 );
4382:     if ( scratch != ( (size_t) ( reg_value[0] & 0x40 ) ) )
4383:     {
4384:         error_code = -1;
4385:     }
4386: }
4387:
4388: if ( error_code < 0 )
4389: {
4390:     printf( "###ERROR: SPI_Status_Read_FIFO_Status: " );
4391:     for ( index = 0; index < 4; index++ ) printf( ", 0x%02X", reg_value[index] );
4392:     printf( "\n" );
4393: }
4394: reg_rx_status = reg_value[0];
4395: #else
4396: switch ( board_id )
4397: {
4398:     case ID_IDI48:
4399:         IO_Read_U8( board_id, __LINE__, IDI_SPI_RX_STATUS, &reg_rx_status );
4400:         break;
4401:     case ID_IDO48:
4402:         IO_Read_U8( board_id, __LINE__, IDO_SPI_RX_STATUS, &reg_rx_status );
4403:         break;
4404: }
4405: #endif
4406:
4407:
4408: *bytes_available = (size_t)( reg_rx_status & SPI_FIFO_SIZE_BIT_WIDTH );
4409: *empty           = (BOOL)( reg_rx_status & 0x40 );
4410: }
4411:
4412: int SPI_Status_Read( int board_id, struct spi_status * status )
4413: {
4414:     uint8_t reg_rx_status;
4415:
4416:     #define FUNCTION_NAME__ "SPI_Status_Read"
4417:     #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
4418:         IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
4419:     #endif
4420:     #undef FUNCTION_NAME__
4421:
4422:     if ( SPI_IsNotPresent( board_id ) ) return -EC_SPI_NOT_FOUND;
4423:
4424:     switch ( board_id )
4425:     {
4426:         case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_RX_STATUS, &reg_rx_status );
break;
4427:         case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_RX_STATUS, &reg_rx_status );
break;
4428:     }
4429:
4430:     status->fifo_count = (int)( reg_rx_status & 0x1F );
4431:     status->full      = (BOOL)( reg_rx_status & 0x80 );
4432:     status->fifo_size = (int) SPI_FIFO_SIZE;
4433:     status->empty     = (BOOL)( reg_rx_status & 0x40 );
4434:     status->tx_status = false;
4435:
4436:     return SUCCESS;
4437: }

```

```
4454:
4462: BOOL SPI_Status_Write_FIFO_Is_Not_Empty( int board_id )
4463: {
4464:     uint8_t reg_tx_status;
4465:
4466: #define FUNCTION_NAME__ "SPI_Status_Write_FIFO_Is_Not_Empty"
4467: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
4468:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
4469: #endif
4470: #undef FUNCTION_NAME__
4471:
4472:     switch ( board_id )
4473:     {
4474:         case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_TX_STATUS, &reg_tx_status );
break;
4475:         case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_TX_STATUS, &reg_tx_status );
break;
4476:     }
4477:     if ( reg_tx_status & 0x40 ) return false;
4478:     return true;
4479: }
4480:
4488: BOOL SPI_Status_Read_FIFO_Is_Not_Empty( int board_id )
4489: {
4490:     uint8_t reg_rx_status;
4491:
4492: #define FUNCTION_NAME__ "SPI_Status_Read_FIFO_Is_Not_Empty"
4493: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
4494:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
4495: #endif
4496: #undef FUNCTION_NAME__
4497:
4498:     switch ( board_id )
4499:     {
4500:         case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_RX_STATUS, &reg_rx_status );
break;
4501:         case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_RX_STATUS, &reg_rx_status );
break;
4502:     }
4503:     if ( reg_rx_status & 0x40 ) return false;
4504:     return true;
4505: }
4506:
4511: #define SPI_COMMIT_BIT_POSITION 0x01
4512:
4513:
4524: int SPI_Commit( int board_id, uint8_t commit )
4525: {
4526: #define FUNCTION_NAME__ "SPI_Commit"
4527: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
4528:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
4529: #endif
4530: #undef FUNCTION_NAME__
4531:
4532:     if ( SPI_IsNotPresent( board_id ) ) return -EC_SPI_NOT_FOUND;
4533:     commit = commit & SPI_COMMIT_BIT_POSITION;
4534:     switch ( board_id )
4535:     {
4536:         case ID_IDI48: IO_Write_U8( board_id, __LINE__, IDI_SPI_COMMIT, commit ); break;
4537:         case ID_IDO48: IO_Write_U8( board_id, __LINE__, IDO_SPI_COMMIT, commit ); break;
4538:     }
4539:     return SUCCESS;
4540: }
4541:
```

```

4552: int SPI_Commit_Get( int board_id, uint8_t * commit )
4553: {
4554:     uint8_t value;
4555:
4556: #define FUNCTION_NAME__ "SPI_Commit_Get"
4557: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
4558:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
4559: #endif
4560: #undef FUNCTION_NAME__
4561:
4562:
4563: if ( SPI_IsNotPresent( board_id ) ) return -EC_SPI_NOT_FOUND;
4564: switch ( board_id )
4565: {
4566:     case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_COMMIT, &value ); break;
4567:     case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_COMMIT, &value ); break;
4568: }
4569: *commit = value & SPI_COMMIT_BIT_POSITION;
4570: return SUCCESS;
4571: }
4572:
4579: BOOL SPI_Commit_Is_Inactive( int board_id )
4580: {
4581:     uint8_t value;
4582:
4583: #define FUNCTION_NAME__ "SPI_Commit_Is_Inactive"
4584: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
4585:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
4586: #endif
4587: #undef FUNCTION_NAME__
4588:
4589: switch ( board_id )
4590: {
4591:     case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_COMMIT, &value ); break;
4592:     case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_COMMIT, &value ); break;
4593: }
4594:
4595: if ( 0 == (value & SPI_COMMIT_BIT_POSITION) ) return true;
4596: return false;
4597: }
4598:
4614: int SPI_FIFO_Write( int board_id, const void * buffer, size_t size, size_t count, FILE
* fd_log )
4615: {
4616:     int error_code;
4617:     size_t bytes_available;
4618:     BOOL empty;
4619:     BOOL full;
4620:     size_t index;
4621:     size_t qty_objects;
4622:     size_t qty_bytes;
4623:     int reg_symbol;
4624:
4625: #define FUNCTION_NAME__ "SPI_FIFO_Write"
4626: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
4627:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
4628: #endif
4629: #undef FUNCTION_NAME__
4630:
4631:     error_code = SUCCESS;
4632:
4633:     if ( (size * count) > SPI_FIFO_SIZE ) return -EC_PARAMETER;
4634:
4635:     if ( SPI_IsNotPresent( board_id ) ) return -EC_SPI_NOT_FOUND;

```

```
4636:  
4637: SPI_Status_Write_FIFO_Status( board_id, &full, &empty, &bytes_available );  
4638:  
4639: qty_objects = bytes_available / size;  
4640: if ( count < qty_objects ) qty_objects = count;  
4641:  
4642: qty_bytes = qty_objects * size;  
4643:  
4644:  
4645:  
4646:  
4647: switch ( board_id )  
4648: {  
4649:   case ID_IDI48: reg_symbol = (int) IDI_SPI_DATA; break;  
4650:   case ID_IDO48: reg_symbol = (int) IDO_SPI_DATA; break;  
4651: }  
4652: for ( index = 0; index < qty_bytes; index++ ) IO_Write_U8( board_id, __LINE__,  
reg_symbol, ((uint8_t *) buffer)[index] );  
4653:  
4654: if ( NULL != fd_log )  
4655: {  
4656:   error_code = fwrite( buffer, size, qty_objects, fd_log );  
4657: }  
4658:  
4659: if ( SUCCESS == error_code ) error_code = (int) qty_objects ;  
4660: return error_code;  
4661: }  
4662:  
4678: int SPI_FIFO_Read( int board_id, const void * buffer, size_t size, size_t count, FILE *  
fd_log )  
4679: {  
4680:   int error_code;  
4681:   size_t bytes_available;  
4682:   BOOL empty;  
4683:   size_t index;  
4684:   size_t qty_objects;  
4685:   size_t qty_bytes;  
4686:   int reg_symbol;  
4687:  
4688: #define FUNCTION_NAME__ "SPI_FIFO_Read"  
4689: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )  
4690:   IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );  
4691: #endif  
4692: #undef FUNCTION_NAME__  
4693:  
4694:   error_code = SUCCESS;  
4695:  
4696:   if ( (size * count) > SPI_FIFO_SIZE ) return -EC_PARAMETER;  
4697:  
4698:   if ( SPI_IsNotPresent( board_id ) ) return -EC_SPI_NOT_FOUND;  
4699:  
4700:   SPI_Status_Read_FIFO_Status( board_id, &empty, &bytes_available );  
4701:  
4702:   qty_objects = bytes_available / size;  
4703:   if ( count < qty_objects ) qty_objects = count;  
4704:  
4705:   qty_bytes = qty_objects * size;  
4706:  
4707:   switch ( board_id )  
4708:   {  
4709:     case ID_IDI48: reg_symbol = (int) IDI_SPI_DATA; break;  
4710:     case ID_IDO48: reg_symbol = (int) IDO_SPI_DATA; break;  
4711:   }  
4712:   for ( index = 0; index < qty_bytes; index++ ) IO_Read_U8( board_id, __LINE__,
```

```
reg_symbol, &(((uint8_t *) buffer)[index]) );
4713:
4714: if ( NULL != fd_log )
4715: {
4716:     error_code = fwrite( buffer, size, qty_objects, fd_log );
4717: }
4718:
4719: if ( SUCCESS == error_code ) error_code = ( (int) qty_objects );
4720: return error_code;
4721: }
4722:
4723:
4727: static void SPI_Data_Write_Read_Helper_Read( int      board_id,
4728:                                              size_t    object_size,
4729:                                              size_t    object_total_count,
4730:                                              BOOL      active_rx,
4731:                                              size_t *   index_rx,
4732:                                              const void * rx_buffer
4733:                                         )
4734: {
4735:     BOOL rx_empty;
4736:     int  reg_symbol;
4737:     size_t rx_bytes_available;
4738:     size_t object_qty;
4739:     size_t object_remaining;
4740:     size_t index_object;
4741:     size_t index;
4742:     size_t index_byte;
4743:     uint8_t bit_bucket;
4744:
4745: #if defined( SPI_PRINT_XFR_DEBUG )
4746:     printf( " SPI_Data_Write_Read_Helper_Read: " );
4747: #endif
4748:
4749: #define FUNCTION_NAME__ "SPI_Data_Write_Read_Helper_Read"
4750: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
4751:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
4752: #endif
4753: #undef FUNCTION_NAME__
4754:
4755:
4756: SPI_Status_Read_FIFO_Status( board_id, &rx_empty, &rx_bytes_available );
4757: if ( rx_bytes_available >= object_size )
4758: {
4759:     object_qty = rx_bytes_available / object_size;
4760:     object_remaining = object_total_count - *index_rx;
4761:     if ( object_qty > object_remaining )
4762:     {
4763:         object_qty = object_total_count - *index_rx;
4764:     }
4765:
4766:     switch ( board_id )
4767:     {
4768:         case ID_IDI48: reg_symbol = (int) IDI_SPI_DATA; break;
4769:         case ID_IDO48: reg_symbol = (int) IDO_SPI_DATA; break;
4770:     }
4771:
4772:     if ( active_rx )
4773:     {
4774:         for ( index_object = 0; index_object < object_qty; index_object++ )
4775:         {
4776:             index_byte = (*index_rx) * object_size;
4777:             for ( index = 0; index < object_size; index++ )
4778:             {
```

```
4779:     IO_Read_U8( board_id, __LINE__, reg_symbol, &(((uint8_t *) rx_buffer)[index_byte + index]) );
4780: #if defined( SPI_PRINT_XFR_DEBUG )
4781: global_internal_rx_byte_count++;
4782: #endif
4783: }
4784:     *index_rx = *index_rx + 1;
4785: }
4786: }
4787: else
4788: {
4789:     for ( index_object = 0; index_object < object_qty; index_object++ )
4790:     {
4791:         index_byte = (*index_rx) * object_size;
4792:         for ( index = 0; index < object_size; index++ )
4793:         {
4794:             IO_Read_U8( board_id, __LINE__, reg_symbol, &bit_bucket );
4795: #if defined( SPI_PRINT_XFR_DEBUG )
4796: global_internal_rx_byte_count++;
4797: #endif
4798:         }
4799:         *index_rx = *index_rx + 1;
4800:     }
4801: }
4802: #if defined( SPI_PRINT_XFR_DEBUG )
4803: printf( "object_qty=%u, rx_bytes_available=%u, object_remaining=%u, object_size=%u",
object_qty, rx_bytes_available, object_remaining, object_size );
4804: #endif
4805: }
4806: #if defined( SPI_PRINT_XFR_DEBUG )
4807: printf( ", index_rx=%u\n", *index_rx );
4808: #endif
4809:
4810: }
4811:
4812: static void SPI_Data_Write_Read_Helper_Commit( int      board_id,
4813:                                                 size_t    object_size,
4814:                                                 SPI_CSB_ENUM csb,
4815:                                                 BOOL      commit_ready,
4816:                                                 BOOL *    commit_valid
4817: )
4818:
4819:
4820:
4821:
4822:
4823: {
4824:
4825:
4826:
4827:     (void) object_size;
4828:
4829: #define FUNCTION_NAME__ "SPI_Data_Write_Read_Helper_Commit"
4830: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
4831:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
4832: #endif
4833: #undef FUNCTION_NAME__
4834:
4835:
4836:
4837:     if ( true == commit_ready )
4838:     {
4839:         if ( false == *commit_valid )
4840:         {
4841:             if ( CSB_SOFTWARE != csb ) SPI_Commit( board_id, 0xFF );
4842:         }
4843: #if defined( SPI_PRINT_DEBUG )
4844: printf( "SPI_Data_Write_Read_Helper_Commit: commit_valid = true\n" );
4845: #endif
```

```
4846:     *commit_valid = true;
4847: }
4848: else
4849: {
4850:     if ( CSB_BUFFER == csb )
4851:     {
4852:         if ( SPI_Commit_Is_Inactive( board_id ) )
4853:         {
4854:             #if defined( SPI_PRINT_DEBUG )
4855:             printf( "SPI_Data_Write_Read_Helper_Commit: commit_valid = false (CSB = BUFFER)\n" );
4856:         #endif
4857:         *commit_valid = false;
4858:     }
4859: }
4860: else
4861: {
4862:     #if defined( SPI_PRINT_DEBUG )
4863:     printf( "SPI_Data_Write_Read_Helper_Commit: commit_valid = false\n" );
4864:     #endif
4865:     *commit_valid = false;
4866: }
4867: }
4868: }
4869:
4873: static void SPI_Data_Write_Read_Helper_Write( int      board_id,
4874:                                                 size_t    object_size,
4875:                                                 size_t    object_total_count,
4876:                                                 SPI_CSB_ENUM csb,
4877:                                                 BOOL     active_tx,
4878:                                                 BOOL     commit_valid,
4879:                                                 BOOL *    commit_ready,
4880:                                                 size_t *   index_tx,
4881:                                                 const void * tx_buffer
4882:                                         )
4883: {
4884:     BOOL     tx_full;
4885:     BOOL     tx_empty;
4886:     int      reg_symbol;
4887:     size_t    tx_bytes_available;
4888:     size_t    object_qty;
4889:     size_t    object_remaining;
4890:     size_t    index_object;
4891:     size_t    index;
4892:     size_t    index_byte;
4893:
4894:     #if defined( SPI_PRINT_XFR_DEBUG )
4895:     int      reg_tx_status, reg_rx_status;
4896:     uint8_t   reg_tx_status_value, reg_rx_status_value;
4897:     #endif
4898:
4899:     const uint8_t  tx_dummy_value = 0x00;
4900:
4901:
4902:     #if defined( SPI_PRINT_XFR_DEBUG )
4903:     printf( " SPI_Data_Write_Read_Helper_Write: " );
4904:     #endif
4905:
4906:     #define FUNCTION_NAME__ "SPI_Data_Write_Read_Helper_Write"
4907:     #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
4908:         IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
4909:     #endif
4910:     #undef FUNCTION_NAME__
4911:
4912:     if ( false == commit_valid )
```

```
4913: {
4914: #if defined( SPI_PRINT_XFR_DEBUG )
4915: printf( " GO" );
4916: #endif
4917: if ( false == *commit_ready )
4918: {
4919: #if defined( SPI_PRINT_XFR_DEBUG )
4920: printf( ", FILL" );
4921: #endif
4922:
4923: SPI_Status_Write_FIFO_Status( board_id, &tx_full, &tx_empty, &tx_bytes_available );
4924: if ( tx_bytes_available >= object_size )
4925: {
4926: object_qty = tx_bytes_available / object_size;
4927: object_remaining = object_total_count - *index_tx;
4928: if ( object_qty > object_remaining )
4929: {
4930: object_qty = object_total_count - *index_tx;
4931: }
4932:
4933: if ( CSB_BUFFER == csb )
4934: {
4935: if ( object_qty >= 1 ) object_qty = 1;
4936: }
4937: #if defined( SPI_PRINT_DEBUG )
4938: printf( " SPI_Data_Write_Read_Helper_Write: TX object_qty = %d\n", (int) object_qty );
4939: #endif
4940: switch ( board_id )
4941: {
4942: case ID_IDI48:
4943: reg_symbol = (int) IDI_SPI_DATA;
4944: #if defined( SPI_PRINT_XFR_DEBUG )
4945: reg_tx_status = IDI_SPI_TX_STATUS;
4946: reg_rx_status = IDI_SPI_RX_STATUS;
4947: #endif
4948: break;
4949: case ID_IDO48:
4950: reg_symbol = (int) IDO_SPI_DATA;
4951: #if defined( SPI_PRINT_XFR_DEBUG )
4952: reg_tx_status = IDO_SPI_TX_STATUS;
4953: reg_rx_status = IDO_SPI_RX_STATUS;
4954: #endif
4955: break;
4956: }
4957:
4958: if ( active_tx )
4959: {
4960: #if defined( SPI_PRINT_DEBUG )
4961: printf( " SPI_Data_Write_Read_Helper_Write: TX active\n" );
4962: #endif
4963:
4964: #if defined( SPI_PRINT_XFR_DEBUG )
4965: printf( ", DSTATUS:" );
4966: #endif
4967: for ( index_object = 0; index_object < object_qty; index_object++ )
4968: {
4969: #if defined( SPI_PRINT_DEBUG )
4970: printf( " SPI_Data_Write_Read_Helper_Write: TX bytes:" );
4971: #endif
4972: index_byte = *index_tx * object_size;
4973: for ( index = 0; index < object_size; index++ )
4974: {
4975: #if defined( SPI_PRINT_DEBUG )
4976: printf( " 0x%02X", ((uint8_t *) tx_buffer)[index_byte + index] );
```

```

4977: #endif
4978:     IO_Write_U8( board_id, __LINE__, reg_symbol, ((uint8_t *) tx_buffer)[index_byte
+ index] );
4979:
4980:
4981: #if defined( SPI_PRINT_XFR_DEBUG )
4982: global_internal_tx_byte_count++;
4983: # if defined( __MSDOS__ )
4984: delay( 200 );
4985: #endif
4986: IO_Read_U8( board_id, __LINE__, reg_tx_status, &reg_tx_status_value );
4987: IO_Read_U8( board_id, __LINE__, reg_rx_status, &reg_rx_status_value );
4988: printf( " [0x%02X.0x%02X]", reg_tx_status_value, reg_rx_status_value );
4989: #endif
4990:
4991:
4992:     }
4993:     *index_tx = *index_tx + 1;
4994: }
4995: #if defined( SPI_PRINT_DEBUG )
4996: printf( "\n" );
4997: #endif
4998: }
4999: else
5000: {
5001: #if defined( SPI_PRINT_DEBUG )
5002: printf( " SPI_Data_Write_Read_Helper_Write: TX inactive\n" );
5003: #endif
5004:
5005:
5006: #if defined( SPI_PRINT_XFR_DEBUG )
5007: printf( ", NSTATUS:" );
5008: #endif
5009:     for ( index_object = 0; index_object < object_qty; index_object++ )
5010:     {
5011:
5012:         for ( index = 0; index < object_size; index++ )
5013:         {
5014:             IO_Write_U8( board_id, __LINE__, reg_symbol, tx_dummy_value );
5015:
5016: #if defined( SPI_PRINT_XFR_DEBUG )
5017: global_internal_tx_byte_count++;
5018: # if defined( __MSDOS__ )
5019: delay( 200 );
5020: #endif
5021: IO_Read_U8( board_id, __LINE__, reg_tx_status, &reg_tx_status_value );
5022: IO_Read_U8( board_id, __LINE__, reg_rx_status, &reg_rx_status_value );
5023: printf( " [0x%02X.0x%02X]", reg_tx_status_value, reg_rx_status_value );
5024: #endif
5025:
5026:     }
5027:     *index_tx = *index_tx + 1;
5028: }
5029:
5030: }
5031: *commit_ready = true;
5032: #if defined( SPI_PRINT_XFR_DEBUG )
5033: printf( ", object_qty=%u, tx_bytes_available=%u, object_remaining=%u, object_size=%u",
object_qty, tx_bytes_available, object_remaining, object_size );
5034: printf( ", index_tx=%u", *index_tx );
5035: #endif
5036:
5037: #if defined( SPI_PRINT_DEBUG )
5038: printf( "SPI_Data_Write_Read_Helper_Write TX request\n" );

```

```
5039: #endif
5040: }
5041: else
5042: {
5043:
5044: printf( " , FULL" );
5045: }
5046: }
5047: else
5048: {
5049:
5050: printf( " , FULL" );
5051: }
5052: }
5053: else
5054: {
5055: #if defined( SPI_PRINT_DEBUG )
5056: printf( "SPI_Data_Write_Read_Helper_Write TX ack\n" );
5057: #endif
5058: #if defined( SPI_PRINT_XFR_DEBUG )
5059: printf( " ACK" );
5060: #endif
5061: *commit_ready = false;
5062: }
5063: #if defined( SPI_PRINT_XFR_DEBUG )
5064: printf( "\n" );
5065: #endif
5066: }
5067:
5091: int SPI_Data_Write_Read( int board_id,
5092:                         size_t object_size,
5093:                         size_t tx_object_count,
5094:                         const void * tx_buffer,
5095:                         size_t rx_object_count,
5096:                         const void * rx_buffer
5097:                     )
5098: {
5099:     int error_code;
5100:     SPI_CSB_ENUM csb;
5101:     size_t rx_bytes_available;
5102:     BOOL rx_empty;
5103:     size_t tx_bytes_available;
5104:     BOOL active_tx;
5105:     BOOL active_rx;
5106:     BOOL tx_empty;
5107:     BOOL tx_full;
5108:     size_t index_tx;
5109:     size_t index_rx;
5110:
5111:
5112:
5113:
5114:
5115:     BOOL commit_valid;
5116:     BOOL commit_ready;
5117:     uint8_t bit_bucket;
5118:
5119:
5120: #if defined( SPI_PRINT_DEBUG )
5121: #if defined( __MSDOS__ )
5122:     static size_t internal_count = 0;
5123: #endif
5124: #endif
5125:
```

```
5126: struct board_dataset * dataset = ( struct board_dataset * )
board_definition[board_id].dataset;
5127:
5128: #define FUNCTION_NAME__ "SPI_Data_Write_Read"
5129: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
5130: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
5131: #endif
5132: #undef FUNCTION_NAME__
5133:
5134:
5135: #if defined( SPI_PRINT_DEBUG )
5136: printf( "*** SPI_Data_Write_Read: Entry.\n" );
5137: #endif
5138: #if defined( SPI_PRINT_XFR_DEBUG )
5139: printf( "*** SPI_Data_Write_Read: Entry.\n" );
5140: #endif
5141:
5142: index_tx = 0;
5143: index_rx = 0;
5144:
5145:
5146:
5147:
5148: if ( ( NULL == tx_buffer ) && ( NULL == rx_buffer ) )
5149: {
5150:     error_code = SUCCESS;
5151:     goto SPI_Data_Write_Read_Error_Exit;
5152: }
5153: if ( SPI_IsNotPresent( board_id ) )
5154: {
5155:     error_code = -EC_SPI_NOT_FOUND;
5156:     goto SPI_Data_Write_Read_Error_Exit;
5157: }
5158:
5159:
5160: if ( ( NULL != tx_buffer ) && ( NULL != rx_buffer ) )
5161: {
5162:     if ( tx_object_count != rx_object_count )
5163:     {
5164:         error_code = -EC_SPI_BUFFER_MISMATCH;
5165:         goto SPI_Data_Write_Read_Error_Exit;
5166:     }
5167: }
5168: if ( NULL != tx_buffer )
5169: {
5170:     active_tx = true;
5171: }
5172: else
5173: {
5174:     tx_object_count = rx_object_count;
5175:     active_tx      = false;
5176: }
5177: if ( NULL != rx_buffer )
5178: {
5179:     active_rx      = true;
5180: }
5181: else
5182: {
5183:     rx_object_count = tx_object_count;
5184:     active_rx      = false;
5185: }
5186:
5187:
5188: SPI_Configuration_Chip_Select_Behavior_Get( board_id, &csb );
```

```
5189:  
5190: if ( CSB_SOFTWARE == csb )  
5191: {  
5192:     uint8_t chip_select;  
5193:     if ( 0 == SPI_Commit_Get( board_id, &chip_select ) )  
5194:     {  
5195: #if defined( SPI_PRINT_DEBUG )  
5196:         printf( "SPI_Data_Write_Read: Warning Chip select not active, setting it now\n" );  
5197: #endif  
5198:         SPI_Commit( board_id, 0xFF );  
5199:     }  
5200: }  
5201:  
5202: if ( CSB_uint16_t == csb )  
5203: {  
5204:     if ( object_size & 0x01 )  
5205:     {  
5206:         printf( "SPI_Data_Write_Read: Object Size is set to " );  
5207: #if defined( __MSDOS__ )  
5208:         printf( "%d", object_size );  
5209: #else  
5210:         printf( "%lu", object_size );  
5211: #endif  
5212:         printf( ", and ought to be set to 2\n" );  
5213:         error_code = -EC_SPI_BUFFER_SIZE_ODD;  
5214:         goto SPI_Data_Write_Error_Exit;  
5215:     }  
5216: }  
5217:  
5218: #if defined( SPI_PRINT_DEBUG )  
5219: printf( "SPI_Data_Write_Read:\n" );  
5220: #if defined( __MSDOS__ )  
5221: printf( "    object_size      = %d\n", object_size );  
5222: printf( "    tx_object_count = %d\n", tx_object_count );  
5223: printf( "    rx_object_count = %d\n", rx_object_count );  
5224: printf( "    index_tx        = %d\n", index_tx );  
5225: printf( "    index_rx        = %d\n", index_rx );  
5226: printf( "    tx_buffer        = %p\n", tx_buffer );  
5227: printf( "    rx_buffer        = %p\n", rx_buffer );  
5228: #else  
5229: printf( "    object_size      = %lu\n", object_size );  
5230: printf( "    tx_object_count = %lu\n", tx_object_count );  
5231: printf( "    rx_object_count = %lu\n", rx_object_count );  
5232: printf( "    index_tx        = %lu\n", index_tx );  
5233: printf( "    index_rx        = %lu\n", index_rx );  
5234: printf( "    tx_buffer        = %p\n", tx_buffer );  
5235: printf( "    rx_buffer        = %p\n", rx_buffer );  
5236: #endif  
5237: printf( "    active_tx       = %s\n", active_tx ? "true" : "false" );  
5238: printf( "    active_rx       = %s\n", active_rx ? "true" : "false" );  
5239:  
5240: if ( NULL != tx_buffer )  
5241: {  
5242:     size_t count = object_size * tx_object_count;  
5243:     if ( count > HEX_DUMP_BYTES_PER_LINE ) count = HEX_DUMP_BYTES_PER_LINE;  
5244:     Hex_Dump_Line( 0, count, (uint8_t *) tx_buffer, stdout );  
5245: }  
5246: #endif  
5247:  
5248:  
5249: if ( CSB_BUFFER == csb )  
5250: {  
5253:     if ( object_size > SPI_FIFO_SIZE )  
5254:     {
```

```

5255:     error_code = -EC_SPI_OBJECT_SIZE;
5256:     goto SPI_Data_Write_Read_Error_Exit;
5257: }
5258: }
5259:
5260:
5261:
5262:
5263: SPI_Status_Write_FIFO_Status( board_id, &tx_full, &tx_empty, &tx_bytes_available );
5264: if ( ( false == tx_empty ) && ( CSB_SOFTWARE != csb ) ) SPI_Commit( board_id, 0xFF );
5265: do
5266: {
5267:     error_code = SUCCESS;
5268:     if ( dataset->quit_application ) error_code = -EC_APP_TERMINATE_CTRLC;
5269:     if ( Character_Get( NULL ) ) error_code = -EC_APP_TERMINATE;
5270:     if ( SUCCESS != error_code )
5271:     {
5272:         printf( "SPI_Data_Write_Read initial empty tx\n" );
5273:         goto SPI_Data_Write_Read_Error_Exit;
5274:     }
5275:     SPI_Status_Write_FIFO_Status( board_id, &tx_full, &tx_empty, &tx_bytes_available );
5276: } while ( false == tx_empty );
5277:
5278:
5279: do
5280: {
5281:     error_code = SUCCESS;
5282:     if ( dataset->quit_application ) error_code = -EC_APP_TERMINATE_CTRLC;
5283:     if ( Character_Get( NULL ) ) error_code = -EC_APP_TERMINATE;
5284:     if ( SUCCESS != error_code )
5285:     {
5286:         printf( "SPI_Data_Write_Read initial empty rx\n" );
5287:         goto SPI_Data_Write_Read_Error_Exit;
5288:     }
5289:     SPI_Status_Read_FIFO_Status( board_id, &rx_empty, &rx_bytes_available );
5290:     if ( false == rx_empty )
5291:     {
5292:         switch ( board_id )
5293:         {
5294:             case ID_IDI48: IO_Read_U8( board_id, __LINE__, IDI_SPI_DATA, &bit_bucket ); break;
5295:             case ID_IDO48: IO_Read_U8( board_id, __LINE__, IDO_SPI_DATA, &bit_bucket ); break;
5296:         }
5297:     }
5298: } while ( false == rx_empty );
5299:
5300: index_tx      = 0;
5301: index_rx      = 0;
5302: commit_valid = false;
5303: commit_ready = false;
5304: while ( index_tx < tx_object_count )
5305: {
5306:     error_code = SUCCESS;
5307:     if ( dataset->quit_application ) error_code = -EC_APP_TERMINATE_CTRLC;
5308:     if ( Character_Get( NULL ) ) error_code = -EC_APP_TERMINATE;
5309:     if ( SUCCESS != error_code )
5310:     {
5311:         printf( "SPI_Data_Write_Read main loop\n" );
5312:         goto SPI_Data_Write_Read_Error_Exit;
5313:     }
5314:
5315: #if defined( SPI_PRINT_XFR_DEBUG )
5316: # if defined( __MSDOS__ )
5317:     delay( 250 );
5318:
```

```
5319: SPI_Data_Write_Read_Helper_Read( board_id, object_size, tx_object_count, active_rx,
&index_rx, rx_buffer );
5320: delay( 250 );
5321: # endif
5322: #endif
5323:
5324: #if defined( SPI_PRINT_DEBUG )
5325: # if defined( __MSDOS__ )
5326: internal_count++;
5327: printf( "SPI_Data_Write_Read: LOOP: internal_count = %u\n", internal_count );
5328: printf( " object_size      = %d\n", object_size );
5329: printf( " tx_object_count   = %d\n", tx_object_count );
5330: printf( " rx_object_count   = %d\n", rx_object_count );
5331: printf( " index_tx          = %d\n", index_tx );
5332: printf( " index_rx          = %d\n", index_rx );
5333: printf( " tx_buffer         = %p\n", tx_buffer );
5334: printf( " rx_buffer         = %p\n", rx_buffer );
5335: delay( 1000 );
5336: # endif
5337: #endif
5338:
5339:
5342: SPI_Data_Write_Read_Helper_Read( board_id, object_size, tx_object_count, active_rx,
&index_rx, rx_buffer );
5343:
5344:
5345: #if defined( SPI_PRINT_DEBUG )
5346: # if defined( __MSDOS__ )
5347: if ( index_rx != index_tx )
5348: {
5349:     error_code = -EC_SPI_BUFFER_MISMATCH;
5350:     printf( "SPI_Data_Write_Read: MISMATCH: internal_count = %u\n", internal_count );
5351:
5352: #if defined( __MSDOS__ )
5353:     printf( " object_size      = %d\n", object_size );
5354:     printf( " tx_object_count   = %d\n", tx_object_count );
5355:     printf( " rx_object_count   = %d\n", rx_object_count );
5356:     printf( " index_tx          = %d\n", index_tx );
5357:     printf( " index_rx          = %d\n", index_rx );
5358:     printf( " tx_buffer         = %p\n", tx_buffer );
5359:     printf( " rx_buffer         = %p\n", rx_buffer );
5360: #else
5361:     printf( " object_size      = %lu\n", object_size );
5362:     printf( " tx_object_count   = %lu\n", tx_object_count );
5363:     printf( " rx_object_count   = %lu\n", rx_object_count );
5364:     printf( " index_tx          = %lu\n", index_tx );
5365:     printf( " index_rx          = %lu\n", index_rx );
5366:     printf( " tx_buffer         = %p\n", tx_buffer );
5367:     printf( " rx_buffer         = %p\n", rx_buffer );
5368: #endif
5369:     printf( " active_tx          = %s\n", active_tx ? "true" : "false" );
5370:     printf( " active_rx          = %s\n", active_rx ? "true" : "false" );
5371:
5372:     goto SPI_Data_Write_Read_Error_Exit;
5373: }
5374: # endif
5375: #endif
5376:
5377:
5378: SPI_Data_Write_Read_Helper_Commit( board_id, object_size, csb, commit_ready,
&commit_valid );
5379:
5380:
5383: SPI_Data_Write_Read_Helper_Write( board_id,
```

```

5384:         object_size,
5385:         tx_object_count,
5386:         csb,
5387:         active_tx,
5388:         commit_valid,
5389:         &commit_ready,
5390:         &index_tx,
5391:         tx_buffer
5392:     );
5393:
5394:
5395:
5396:
5397:     SPI_Data_Write_Read_Helper_Commit( board_id, object_size, csb, commit_ready,
&commit_valid );
5398:
5399: }
5400: #if defined( SPI_PRINT_DEBUG )
5401: printf( " SPI_Data_Write_Read: Post write operations\n" );
5402: #endif
5403:
5404:
5405: do
5406: {
5407:     error_code = SUCCESS;
5408:     if ( dataset->quit_application ) error_code = -EC_APP_TERMINATE_CTRLC;
5409:     if ( Character_Get( NULL ) ) error_code = -EC_APP_TERMINATE;
5410:     if ( SUCCESS != error_code )
5411:     {
5412:         printf( "SPI_Data_Write_Read status ending portion\n" );
5413:         goto SPI_Data_Write_Error_Exit;
5414:     }
5415:     SPI_Data_Write_Read_Helper_Commit( board_id, object_size, csb, commit_ready,
&commit_valid );
5416:     SPI_Status_Write_FIFO_Status( board_id, &tx_full, &tx_empty, &tx_bytes_available );
5417: } while ( false == tx_empty );
5418:
5419: #if defined( SPI_PRINT_DEBUG )
5420: printf( " SPI_Data_Write_Read: Post Read Begin\n" );
5421: #endif
5422:
5423:
5424: while ( index_rx != index_tx )
5425: {
5426:     error_code = SUCCESS;
5427:     if ( dataset->quit_application ) error_code = -EC_APP_TERMINATE_CTRLC;
5428:     if ( Character_Get( NULL ) ) error_code = -EC_APP_TERMINATE;
5429:     if ( SUCCESS != error_code )
5430:     {
5431:         printf( "SPI_Data_Write_Read status ending receive loop\n" );
5432:         goto SPI_Data_Write_Error_Exit;
5433:     }
5434:     SPI_Data_Write_Read_Helper_Read( board_id, object_size, tx_object_count, active_rx,
&index_rx, rx_buffer );
5435: }
5436: return SUCCESS;
5437:
5438: SPI_Data_Write_Error_Exit:
5439:
5440:
5441: #if defined( SPI_PRINT_XFR_DEBUG )
5442: printf( "SPI_Data_Write_Read:\n" );
5443: #if defined( __MSDOS__ )
5444: printf( "    object_size      = %d\n", object_size );

```

```
5445: printf( " tx_object_count      = %d\n", tx_object_count );
5446: printf( " rx_object_count      = %d\n", rx_object_count );
5447: printf( " index_tx            = %d\n", index_tx );
5448: printf( " index_rx            = %d\n", index_rx );
5449: printf( " tx_buffer           = %p\n", tx_buffer );
5450: printf( " rx_buffer           = %p\n", rx_buffer );
5451: #else
5452: printf( " object_size          = %lu\n", object_size );
5453: printf( " tx_object_count      = %lu\n", tx_object_count );
5454: printf( " rx_object_count      = %lu\n", rx_object_count );
5455: printf( " index_tx            = %lu\n", index_tx );
5456: printf( " index_rx            = %lu\n", index_rx );
5457: printf( " tx_buffer           = %p\n", tx_buffer );
5458: printf( " rx_buffer           = %p\n", rx_buffer );
5459: #endif
5460: printf( " active_tx            = %s\n", active_tx ? "true" : "false" );
5461: printf( " active_rx            = %s\n", active_rx ? "true" : "false" );
5462:
5463: printf( " global_internal_tx_byte_count = %u\n", global_internal_tx_byte_count );
5464: printf( " global_internal_rx_byte_count = %u\n", global_internal_rx_byte_count );
5465: #endif
5466:
5467:
5468: #define FUNCTION_NAME__ "SPI_Data_Write_Read"
5469: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
5470: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, error_code );
5471: #endif
5472: #undef FUNCTION_NAME__
5473:
5474: return error_code;
5475: }
5476:
5485:
5491:
5492: int SPI_Data_Write( int      board_id,
5493:                      const void *    tx_buffer,
5494:                      size_t        tx_object_size,
5495:                      size_t        tx_object_count,
5496:                      FILE       *    fd_log
5497:                      )
5498: {
5499:     int error_code;
5500:
5501: #define FUNCTION_NAME__ "SPI_Data_Write"
5502: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
5503: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
5504: #endif
5505: #undef FUNCTION_NAME__
5506:
5507:     error_code = SPI_Data_Write_Read( board_id,
5508:                                         tx_object_size,
5509:                                         tx_object_count,
5510:                                         tx_buffer,
5511:                                         0,
5512:                                         NULL
5513:                                         );
5514:     if ( SUCCESS != error_code ) goto SPI_Data_Write_Error_Exit;
5515:
5516:     if ( NULL != fd_log )
5517:     {
5518:         error_code = fwrite( tx_buffer, tx_object_size, tx_object_count, fd_log );
5519:     }
5520:     if ( error_code < 0 ) goto SPI_Data_Write_Error_Exit;
5521:
```

```
5522:     return SUCCESS;
5523:
5524: SPI_Data_Write_Error_Exit:
5525:
5526: #define FUNCTION_NAME__ "SPI_Data_Write"
5527: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
5528:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, error_code );
5529: #endif
5530: #undef FUNCTION_NAME__
5531:
5532:     return error_code;
5533: }
5534:
5543:
5549: int SPI_Data_Read( int board_id,
5550:                      const void * rx_buffer,
5551:                      size_t rx_object_size,
5552:                      size_t rx_object_count,
5553:                      FILE * fd_log
5554:                      )
5555: {
5556:     int error_code;
5557:
5558: #define FUNCTION_NAME__ "SPI_Data_Read"
5559: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
5560:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, 0 );
5561: #endif
5562: #undef FUNCTION_NAME__
5563:
5564:     error_code = SPI_Data_Write_Read( board_id,
5565:                                         rx_object_size,
5566:                                         0,
5567:                                         NULL,
5568:                                         rx_object_count,
5569:                                         rx_buffer
5570:                                         );
5571:     if ( SUCCESS != error_code ) goto SPI_Data_Read_Error_Exit;;
5572:
5573:     if ( NULL != fd_log )
5574:     {
5575:         error_code = fwrite( rx_buffer, rx_object_size, rx_object_count, fd_log );
5576:     }
5577:     if ( error_code < 0 ) goto SPI_Data_Read_Error_Exit;
5578:
5579:     return SUCCESS;
5580:
5581: SPI_Data_Read_Error_Exit:
5582:
5583: #define FUNCTION_NAME__ "SPI_Data_Read"
5584: #if ( IO_TRACE_USE_SPI_LOGGING == 1 )
5585:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_SPI, board_id, error_code );
5586: #endif
5587: #undef FUNCTION_NAME__
5588:
5589:     return error_code;
5590: }
5591:
5592:
5593:
5594:
5595:
5596:
5597:
5598:
```

```
5599:  
5600:  
5601:  
5602:  
5603:  
5608: enum { FRAM_DENSITY_BYTES = 8192 };  
5609:  
5610:  
5611:  
5620: static int FRAM__Chip_Select_Route_Override( int board_id )  
5621: {  
5622:     struct board_dataset * dataset;  
5623:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;  
5624:  
5625: #define FUNCTION_NAME__ "FRAM__Chip_Select_Route_Override"  
5626: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )  
5627:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );  
5628: #endif  
5629: #undef FUNCTION_NAME__  
5630:  
5631:     SPI_Configuration_Chip_Select_Route_Get( board_id,  
&(dataset->fram_spi_cs_route_backup) );  
5632:  
5633:     SPI_Configuration_Chip_Select_Route_Set( board_id, dataset->fram_cs_default );  
5634:     return SUCCESS;  
5635: }  
5636:  
5644: static int FRAM__Chip_Select_Route_Restore( int board_id )  
5645: {  
5646:     struct board_dataset * dataset;  
5647:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;  
5648:  
5649: #define FUNCTION_NAME__ "FRAM__Chip_Select_Route_Restore"  
5650: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )  
5651:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );  
5652: #endif  
5653: #undef FUNCTION_NAME__  
5654:  
5655:     SPI_Configuration_Chip_Select_Route_Set( board_id, dataset->fram_spi_cs_route_backup );  
5656:     return SUCCESS;  
5657: }  
5658:  
5666: int FRAM__Write_Enable_Latch_Set( int board_id )  
5667: {  
5668:     int error_code;  
5669:     uint8_t tx_buf[1] = { 0x06 };  
5670:  
5671: #define FUNCTION_NAME__ "FRAM__Write_Enable_Latch_Set"  
5672: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )  
5673:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );  
5674: #endif  
5675: #undef FUNCTION_NAME__  
5676:  
5677:     FRAM__Chip_Select_Route_Override( board_id );  
5678:  
5679:     SPI_Configuration_Chip_Select_Behavior_Set( board_id, CSB_BUFFER );  
5680:     error_code = SPI_Data_Write_Read( board_id, 1, 1, tx_buf, 0, NULL );  
5681:     FRAM__Chip_Select_Route_Restore( board_id );  
5682:     return error_code;  
5683: }  
5684:  
5692: int FRAM__Write_Disable( int board_id )  
5693: {  
5694:     int error_code;
```

```
5695: uint8_t tx_buf[1] = { 0x04 };
5696:
5697: #define FUNCTION_NAME__ "FRAM_Write_Disable"
5698: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
5699: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );
5700: #endif
5701: #undef FUNCTION_NAME__
5702:
5703: FRAM_Chip_Select_Route_Override( board_id );
5704:
5705: SPI_Configuration_Chip_Select_Behavior_Set( board_id, CSB_BUFFER );
5706: error_code = SPI_Data_Write_Read( board_id, 1, 1, tx_buf, 0, NULL );
5707: FRAM_Chip_Select_Route_Restore( board_id );
5708: return error_code;
5709: }
5710:
5711: int FRAM_Read_Status_Register( int board_id, uint8_t * status )
5712: {
5713: int error_code;
5714: uint8_t tx_buf[2] = { 0x05, 0x00 };
5715: uint8_t rx_buf[2];
5716:
5717: #define FUNCTION_NAME__ "FRAM_Read_Status_Register"
5718: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
5719: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );
5720: #endif
5721: #undef FUNCTION_NAME__
5722:
5723: FRAM_Chip_Select_Route_Override( board_id );
5724:
5725: SPI_Configuration_Chip_Select_Behavior_Set( board_id, CSB_BUFFER );
5726: error_code = SPI_Data_Write_Read( board_id, 2, 1, tx_buf, 1, rx_buf );
5727: FRAM_Chip_Select_Route_Restore( board_id );
5728: if ( SUCCESS != error_code ) return error_code;
5729: *status = rx_buf[1];
5730:
5731: #if defined( IDI_FRAM_PRINT_DEBUG )
5732: {
5733: int index;
5734: printf( "FRAM_Read_Status_Register:" );
5735: for ( index = 0; index < 2; index++ )
5736: {
5737: printf( " 0x%02X", rx_buf[index] );
5738: }
5739: printf( "\n" );
5740: }
5741: #endif
5742: return SUCCESS;
5743: }
5744:
5745: int FRAM_Write_Status_Register( int board_id, uint8_t status )
5746: {
5747: int error_code;
5748: uint8_t tx_buf[2] = { 0x01, 0x00 };
5749: uint8_t rx_buf[2];
5750:
5751: #define FUNCTION_NAME__ "FRAM_Write_Status_Register"
5752: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
5753: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );
5754: #endif
5755: #undef FUNCTION_NAME__
5756:
5757: FRAM_Write_Enable_Latch_Set( board_id );
5758:
```

```
5775: tx_buf[1] = status;
5776: FRAM__Chip_Select_Route_Override( board_id );
5777:
5778: SPI_Configuration_Chip_Select_Behavior_Set( board_id, CSB_BUFFER );
5779: error_code = SPI_Data_Write_Read( board_id, 2, 1, tx_buf, 1, rx_buf );
5780: FRAM__Chip_Select_Route_Restore( board_id );
5781: if ( SUCCESS != error_code ) return error_code;
5782:
5783: return SUCCESS;
5784: }
5785:
5796: int FRAM__Memory_Read( int board_id, uint16_t address, size_t count, uint8_t * buffer )
5797: {
5798:     int     error_code;
5799:
5800:     uint8_t    tx_buf[3] = { 0x03, 0x00, 0x00 };
5801:
5802:
5803: #define FUNCTION_NAME__ "FRAM__Memory_Read"
5804: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
5805:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );
5806: #endif
5807: #undef FUNCTION_NAME__
5808:
5809:
5810: #if defined( SPI_PRINT_DEBUG )
5811:     printf( "*** FRAM__Memory_Read: Entry.\n" );
5812: #endif
5813:
5814:     tx_buf[1] = (uint8_t)( address >> 8 );
5815:     tx_buf[2] = (uint8_t)( address & 0xFF );
5816:     FRAM__Chip_Select_Route_Override( board_id );
5817:
5818:
5819:
5820:
5821:     error_code = SPI_Configuration_Chip_Select_Behavior_Set( board_id, CSB_SOFTWARE );
5822:     if ( SUCCESS != error_code ) goto FRAM__Memory_Read_Error_Exit;
5823:
5824:     SPI_Commit( board_id, 0xFF );
5825:
5826: #if defined( SPI_PRINT_DEBUG )
5827:     printf( "*** FRAM__Memory_Read: Send FRAM Write Command.\n" );
5828: #endif
5829:
5830:     error_code = SPI_Data_Write_Read( board_id, 3, sizeof( uint8_t ), tx_buf, 0, NULL );
5831:     if ( SUCCESS != error_code ) goto FRAM__Memory_Read_Error_Exit;
5832:
5833: #if defined( SPI_PRINT_DEBUG )
5834:     printf( "*** FRAM__Memory_Read: count=%u\n", count );
5835: #endif
5836:
5837:     error_code = SPI_Data_Write_Read( board_id, sizeof( uint8_t ), 0, NULL, count, buffer );
5838:     if ( SUCCESS != error_code ) goto FRAM__Memory_Read_Error_Exit;
5839:
5840:
5841:     SPI_Commit( board_id, 0x00 );
5842:
5843:
5844:
5845:
5846:     FRAM__Chip_Select_Route_Restore( board_id );
5847:     return SUCCESS;
```

```
5848:
5849: FRAM__Memory_Read_Error_Exit:
5850:   SPI_Commit( board_id, 0x00 );
5851:
5852: #define FUNCTION_NAME__ "FRAM__Memory_Read"
5853: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
5854:   IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, error_code
);
5855: #endif
5856: #undef FUNCTION_NAME__
5857:
5858:   FRAM__Chip_Select_Route_Restore( board_id );
5859:   return error_code;
5860: }
5861:
5872: int FRAM__Memory_Write( int board_id, uint16_t address, size_t count, uint8_t * buffer )
5873: {
5874:   int error_code;
5875:
5876:   uint8_t tx_buf[3] = { 0x02, 0x00, 0x00 };
5877:
5878: #define FUNCTION_NAME__ "FRAM__Memory_Write"
5879: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
5880:   IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );
5881: #endif
5882: #undef FUNCTION_NAME__
5883:
5884:   tx_buf[1] = (uint8_t)( address >> 8 );
5885:   tx_buf[2] = (uint8_t)( address & 0xFF );
5886:
5887:   FRAM__Write_Enable_Latch_Set( board_id );
5888:   FRAM__Chip_Select_Route_OVERRIDE( board_id );
5889:
5890:
5891:
5892:
5893:   error_code = SPI_Configuration_Chip_Select_Behavior_Set( board_id, CSB_SOFTWARE );
5894:   if ( SUCCESS != error_code ) goto FRAM__Memory_Write_Error_Exit;
5895:
5896:   SPI_Commit( board_id, 0xFF );
5897:
5898:   error_code = SPI_Data_Write_Read( board_id, 3, sizeof( uint8_t ), tx_buf, 0, NULL );
5899:   if ( SUCCESS != error_code ) goto FRAM__Memory_Write_Error_Exit;
5900:
5901:   error_code = SPI_Data_Write_Read( board_id, sizeof( uint8_t ), count, buffer, 0, NULL
);
5902:   if ( SUCCESS != error_code ) goto FRAM__Memory_Write_Error_Exit;
5903:
5904:
5905:   SPI_Commit( board_id, 0x00 );
5906:
5907:
5908:
5909:
5910:   FRAM__Chip_Select_Route_Restore( board_id );
5911:   return SUCCESS;
5912:
5913: FRAM__Memory_Write_Error_Exit:
5914:   SPI_Commit( board_id, 0x00 );
5915:
5916: #define FUNCTION_NAME__ "FRAM__Memory_Write"
5917: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
5918:   IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, error_code
);
```

```
5919: #endif
5920: #undef FUNCTION_NAME__
5921:
5922: FRAM__Chip_Select_Route_Restore( board_id );
5923: return error_code;
5924: }
5925:
5926: int FRAM__Read_ID( int board_id, uint32_t * id )
5927: {
5928:     int error_code;
5929:     uint8_t tx_buf[5] = { 0x9F, 0x00, 0x00, 0x00, 0x00 };
5930:     uint8_t rx_buf[5];
5931:
5932: #define FUNCTION_NAME__ "FRAM__Read_ID"
5933: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
5934:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );
5935: #endif
5936: #undef FUNCTION_NAME__
5937:
5938: FRAM__Chip_Select_Route_Override( board_id );
5939: SPI_Configuration_Chip_Select_Behavior_Set( board_id, CSB_BUFFER );
5940: error_code = SPI_Data_Write_Read( board_id, 5, 1, tx_buf, 1, rx_buf );
5941: if ( SUCCESS != error_code ) goto FRAM__Read_ID_Error_Exit;
5942:
5943: {
5944:     uint32_t id_scratch = 0;
5945:     size_t index;
5946:
5947:     for ( index = 1; index < 5; index++ )
5948:     {
5949:         id_scratch = ( id_scratch << 8 ) | ( (uint32_t) rx_buf[index] );
5950:     }
5951:     *id = id_scratch;
5952: }
5953: #if defined( IDI_FRAM_PRINT_DEBUG )
5954: {
5955:     int index;
5956:     printf( "FRAM__Read_ID:" );
5957:     for ( index = 0; index < 5; index++ )
5958:     {
5959:         printf( " 0x%02X", rx_buf[index] );
5960:     }
5961:     printf( "\n" );
5962: }
5963: #endif
5964: FRAM__Chip_Select_Route_Restore( board_id );
5965: return SUCCESS;
5966: FRAM__Read_ID_Error_Exit:
5967:
5968: #define FUNCTION_NAME__ "FRAM__Read_ID"
5969: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
5970:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, error_code
);
5971: #endif
5972: #undef FUNCTION_NAME__
5973:
5974: FRAM__Chip_Select_Route_Restore( board_id );
5975: return error_code;
5976: }
5977:
5978: int FRAM_Set( int board_id, size_t count, uint8_t * buffer )
5979: {
5980:     int error_code;
```

```

6016: size_t block_count;
6017: size_t block_remainder;
6018: uint16_t address;
6019: #if defined( IDI_FRAM_PRINT_DEBUG )
6020: size_t index;
6021: #endif
6022:
6023: struct board_dataset * dataset = ( struct board_dataset * )
board_definition[board_id].dataset;
6024:
6025: #define FUNCTION_NAME__ "FRAM_Set"
6026: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
6027: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );
6028: #endif
6029: #undef FUNCTION_NAME__
6030:
6031:
6032:
6033:
6034: address = 0;
6035: error_code = SUCCESS;
6036:
6037: if ( count > 1 )
6038: {
6039:     block_count = ((size_t) FRAM_DENSITY_BYTES) / count;
6040:     block_remainder = ((size_t) FRAM_DENSITY_BYTES) - ( block_count * count );
6041: }
6042: else
6043: {
6044:     int index;
6045:     const int block_size = FRAM_BLOCK_SIZE;
6046:
6047:     if ( NULL == buffer )
6048:     {
6049:         for ( index = 0; index < block_size; index++ ) dataset->spi.fram_block[index] = 0;
6050:     }
6051:     else
6052:     {
6053:         for ( index = 0; index < block_size; index++ ) dataset->spi.fram_block[index] =
buffer[0];
6054:     }
6055:     block_count = ((size_t) FRAM_DENSITY_BYTES) / block_size;
6056:     block_remainder = ((size_t) FRAM_DENSITY_BYTES) - ( block_count * block_size );
6057:     buffer = dataset->spi.fram_block;
6058:     count = block_size;
6059: }
6060:
6061: #if defined( IDI_FRAM_PRINT_DEBUG )
6062: index = 0;
6063: # if defined( __MSDOS__ )
6064:     printf( "FRAM_Set: count=%d, block_count=%d, block_remainder=%d\n", count,
block_count, block_remainder );
6065: # else
6066:     printf( "FRAM_Set: count=%lu, block_count=%lu, block_remainder=%lu\n", count,
block_count, block_remainder );
6067: # endif
6068: #endif
6069:
6070: while ( block_count > 0 )
6071: {
6072:     error_code = FRAM__Memory_Write( board_id, address, count, buffer );
6073:     if ( SUCCESS != error_code ) goto FRAM_Set_Error_Exit;
6074:     address += (uint16_t) count;
6075:     block_count--;

```

```
6076:  
6077: #if defined( IDI_FRAM_PRINT_DEBUG )  
6078: if ( index < 16 )  
6079: {  
6080: # if defined( __MSDOS__ )  
6081: printf( "FRAM_Set: block_count=%d, address=%d\n", block_count, address );  
6082: # else  
6083: printf( "FRAM_Set: block_count=%lu, block_remainder=%d\n", block_count, address );  
6084: # endif  
6085: index++;  
6086: }  
6087: #endif  
6088:  
6089: }  
6090:  
6091: if ( block_remainder > 0 )  
6092: {  
6093: error_code = FRAM__Memory_Write( board_id, address, block_remainder, buffer );  
6094: if ( SUCCESS != error_code ) goto FRAM_Set_Error_Exit;  
6095: }  
6096:  
6097:  
6098: return SUCCESS;  
6099:  
6100: FRAM_Set_Error_Exit:  
6101:  
6102: #define FUNCTION_NAME__ "FRAM_Set"  
6103: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )  
6104: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, error_code  
);  
6105: #endif  
6106: #undef FUNCTION_NAME__  
6107:  
6108: return error_code;  
6109: }  
6110:  
6111: int FRAM_Report( int board_id, uint16_t address, size_t length, FILE * out )  
6112: {  
6113: int error_code;  
6114: const int block_size = HEX_DUMP_BYTES_PER_LINE;  
6115: size_t block_count;  
6116: size_t block_remainder;  
6117: struct board_dataset * dataset = ( struct board_dataset * )  
board_definition[board_id].dataset;  
6118:  
6119: #define FUNCTION_NAME__ "FRAM_Report"  
6120: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )  
6121: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );  
6122: #endif  
6123: #undef FUNCTION_NAME__  
6124:  
6125: #if defined( SPI_PRINT_XFR_DEBUG )  
6126: fprintf( out, "FRAM using SPI CS channel %d\n", dataset->fram_cs_default ? 1 : 0 );  
6127: #endif  
6128:  
6129:  
6130: block_count = ((size_t) length) / block_size;  
6131: block_remainder = ((size_t) length) - ( block_count * block_size );  
6132:  
6133:  
6134: while ( block_count > 0 )  
6135: {  
6136: error_code = FRAM__Memory_Read( board_id, address, block_size,  
dataset->spi.fram_block );  
6137: if ( SUCCESS != error_code ) goto FRAM_Report_Error_Exit;  
6138: error_code = Hex_Dump_Line( address, block_size, dataset->spi.fram_block, out );  
6139:  
6140:  
6141:
```

```

6142: if ( SUCCESS != error_code ) goto FRAM_Report_Error_Exit;
6143: address += block_size;
6144: block_count--;
6145: }
6146:
6147: if ( block_remainder > 0 )
6148: {
6149:   error_code = FRAM__Memory_Read( board_id, address, block_remainder,
dataset->spi.fram_block );
6150:   if ( SUCCESS != error_code ) goto FRAM_Report_Error_Exit;
6151:   error_code = Hex_Dump_Line( address, block_remainder, dataset->spi.fram_block, out );
6152:   if ( SUCCESS != error_code ) goto FRAM_Report_Error_Exit;
6153: }
6154:
6155: return SUCCESS;
6156:
6157: FRAM_Report_Error_Exit:
6158:
6159: #define FUNCTION_NAME__ "FRAM_Report"
6160: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
6161:   IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, error_code
);
6162: #endif
6163: #undef FUNCTION_NAME__
6164:
6165: return error_code;
6166: }
6167:
6173: int FRAM_Memory_To_File( int board_id, uint16_t address, size_t length, FILE * binary )
6174: {
6175:   int error_code;
6176:   size_t block_count;
6177:   size_t block_remainder;
6178:   struct board_dataset * dataset      = ( struct board_dataset * )
board_definition[board_id].dataset;
6179:
6180: #define FUNCTION_NAME__ "FRAM_Memory_To_File"
6181: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
6182:   IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );
6183: #endif
6184: #undef FUNCTION_NAME__
6185:
6186:   block_count = length / ((size_t) FRAM_BLOCK_SIZE);
6187:   block_remainder = length - ( block_count * ((size_t) FRAM_BLOCK_SIZE) );
6188:
6189:   while ( block_count > 0 )
6190:   {
6191:     error_code = FRAM__Memory_Read( board_id, address, ((size_t) FRAM_BLOCK_SIZE),
dataset->spi.fram_block );
6192:     if ( SUCCESS != error_code ) goto FRAM_Memory_To_File_Error_Exit;
6193:     fwrite( dataset->spi.fram_block, 1, ((size_t) FRAM_BLOCK_SIZE), binary );
6194:     address += FRAM_BLOCK_SIZE;
6195:     block_count--;
6196:   }
6197:
6198:   if ( block_remainder > 0 )
6199:   {
6200:     error_code = FRAM__Memory_Read( board_id, address, block_remainder,
dataset->spi.fram_block );
6201:     if ( SUCCESS != error_code ) goto FRAM_Memory_To_File_Error_Exit;
6202:     fwrite( dataset->spi.fram_block, 1, block_remainder, binary );
6203:   }
6204:   return SUCCESS;
6205:
```

```
6206: FRAM_Memory_To_File_Error_Exit:
6207:
6208: #define FUNCTION_NAME__ "FRAM_Memory_To_File"
6209: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
6210:   IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, error_code
);
6211: #endif
6212: #undef FUNCTION_NAME__
6213:
6214:   return error_code;
6215: }
6216:
6222: int FRAM_File_To_Memory( int board_id, uint16_t address, size_t length, FILE * binary )
6223: {
6224:   int error_code;
6225:   size_t count_read;
6226:   size_t count_total;
6227:   size_t count_actual;
6228:   struct board_dataset * dataset    = ( struct board_dataset * )
board_definition[board_id].dataset;
6229:
6230: #define FUNCTION_NAME__ "FRAM_File_To_Memory"
6231: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
6232:   IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, 0 );
6233: #endif
6234: #undef FUNCTION_NAME__
6235:
6236:   count_total = 0;
6237:   count_read  = FRAM_BLOCK_SIZE;
6238:   if ( 0 == length )
6239:   {
6240:     do
6241:     {
6242:       count_actual = fread( dataset->spi.fram_block, 1, count_read, binary );
6243:       error_code   = FRAM_Memory_Write( board_id, address, count_read,
dataset->spi.fram_block );
6244:       if ( SUCCESS != error_code ) goto FRAM_File_To_Memory_Error_Exit;
6245:       count_total += count_actual;
6246:       if ( count_actual != count_read ) count_read = 0;
6247:     } while ( count_read > 0 );
6248:   }
6249: else
6250: {
6251:   if ( length < count_read ) count_read = length;
6252:   do
6253:   {
6254:     count_actual = fread( dataset->spi.fram_block, 1, count_read, binary );
6255:     error_code   = FRAM_Memory_Write( board_id, address, count_read,
dataset->spi.fram_block );
6256:     if ( SUCCESS != error_code ) goto FRAM_File_To_Memory_Error_Exit;
6257:     count_total += count_actual;
6258:     length      -= count_actual;
6259:     if ( count_actual != count_read ) count_read = 0;
6260:     if ( length < count_read ) count_read = length;
6261:   } while ( count_read > 0 );
6262: }
6263: return SUCCESS;
6264:
6265: FRAM_File_To_Memory_Error_Exit:
6266:
6267: #define FUNCTION_NAME__ "FRAM_File_To_Memory"
6268: #if ( IO_TRACE_USE_FRAM_LOGGING == 1 )
6269:   IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_FRAM, board_id, error_code
);
```

```
6270: #endif
6271: #undef FUNCTION_NAME__
6272:
6273:     return error_code;
6274: }
6275:
6276:
6277:
6278:
6279:
6280:
6281:
6282:
6283: struct command_line
6284: {
6285:     struct command_line * link;
6286:     int (* cmd_fnc )( int argc, char * argv[] );
6287:     char * name;
6288:     char * help;
6289: };
6290:
6291:
6292:
6293:
6294:
6295: struct command_line_board
6296: {
6297:     struct command_line_board * link;
6298:     int (* cmd_fnc )( int board_id, int argc, char * argv[] );
6299:     char * name;
6300:     char * help;
6301: };
6302:
6303:
6304:
6305:
6306:
6307:
6308:
6309:
6310:
6311:
6312:
6313:
6314: static int CMD__SPI_ID( int board_id, int argc, char * argv[] )
6315: {
6316:     uint16_t id;
6317:     (void) argc;
6318:     (void) argv;
6319:
6320:     SPI_ID_Get( board_id, &id );
6321:     printf( "SPI ID: 0x%04X\n", id );
6322:     return SUCCESS;
6323:
6324:
6325: }
6326:
6327:
6328:
6329:
6330:
6331:
6332:
6333: static int CMD__SPI_Config_Get( int board_id, int argc, char * argv[] )
6334: {
6335:     int error_code;
6336:     struct spi_cfg cfg;
6337:     (void) argc;
6338:     (void) argv;
6339:
6340:     error_code = SPI_Configuration_Get( board_id, &cfg );
6341:     if ( SUCCESS != error_code ) return error_code;
6342:
6343:     error_code = SPI_Report_Configuration_Text( &cfg, stdout );
6344:     if ( SUCCESS != error_code ) return error_code;
6345:
6346:
6347:
```

```
6348: printf( "\n" );
6349: return SUCCESS;
6350: }
6351:
6357: static int CMD__SPI_Config_Clock_Hz( int board_id, int argc, char * argv[] )
6358: {
6359:     int error_code;
6360:     double clock_request_hz;
6361:     double clock_actual_hz;
6362:     double error;
6363:     uint16_t hci;
6364:     struct spi_cfg cfg;
6365:
6366:
6367:     error_code = SPI_Configuration_Get( board_id, &cfg );
6368:     if ( SUCCESS != error_code ) return error_code;
6369:
6370:     if ( argc < 1 )
6371:     {
6372:         printf( "SPI CLK: %f hz\n", cfg.clock_hz );
6373:     }
6374:     else
6375:     {
6376:         clock_request_hz = atof( argv[0] );
6377:         error_code = SPI_Calculate_Clock( clock_request_hz, &clock_actual_hz, &error, &hci );
6378:         if ( SUCCESS != error_code ) return error_code;
6379:
6380:
6381:         cfg.clock_hz = clock_actual_hz;
6382:         error_code = SPI_Configuration_Set( board_id, &cfg );
6383:         if ( SUCCESS != error_code ) return error_code;
6384:         printf( "OK\n" );
6385:     }
6386:     return SUCCESS;
6387: }
6388:
6394: static int CMD__SPI_Config_End_Cycle_Delay_Sec( int board_id, int argc, char * argv[] )
6395: {
6396:     int error_code;
6397:     double request_sec;
6398:     double actual_sec;
6399:     double error;
6400:     uint8_t ecd;
6401:     struct spi_cfg cfg;
6402:
6403:
6404:     error_code = SPI_Configuration_Get( board_id, &cfg );
6405:     if ( SUCCESS != error_code ) return error_code;
6406:
6407:     if ( argc < 1 )
6408:     {
6409:         printf( "SPI ECD: %g sec\n", ( cfg.end_delay_ns * 1.0e-9 ) );
6410:     }
6411:     else
6412:     {
6413:         request_sec = atof( argv[0] );
6414:         error_code = SPI_Calculate_End_Cycle_Delay( SPI_Calculate_Half_Clock_Interval_Sec(
cfg.half_clock_interval ),
6415:                                         request_sec,
6416:                                         &actual_sec,
6417:                                         &error,
6418:                                         &ecd
6419:                                         );
6420:         if ( SUCCESS != error_code ) return error_code;
```

```

6421:     cfg.end_delay_ns = actual_sec * 1.0e9;
6422:
6423:     error_code = SPI_Configuration_Set( board_id, &cfg );
6424:     if ( SUCCESS != error_code ) return error_code;
6425:     printf( "OK\n" );
6426: }
6427: return SUCCESS;
6428: }
6429:
6430:
6431: static int CMD__SPI_Config_Mode( int board_id, int argc, char * argv[] )
6432: {
6433:     int error_code;
6434:     int mode;
6435:     struct spi_cfg cfg;
6436:
6437:     error_code = SPI_Configuration_Get( board_id, &cfg );
6438:     if ( SUCCESS != error_code ) return error_code;
6439:
6440:     if ( argc < 1 )
6441:     {
6442:         if ( (false == cfg.sclk_polarity) && (false == cfg.sclk_phase) ) mode = 0;
6443:         else if ( (false == cfg.sclk_polarity) && (true == cfg.sclk_phase) ) mode = 1;
6444:         else if ( (true == cfg.sclk_polarity) && (false == cfg.sclk_phase) ) mode = 2;
6445:         else if ( (true == cfg.sclk_polarity) && (true == cfg.sclk_phase) ) mode = 3;
6446:         printf( "SPI MODE: %d\n", mode );
6447:     }
6448:     else
6449:     {
6450:         mode = (int) strtol( argv[0], NULL, 0 );
6451:         switch ( mode )
6452:         {
6453:             case 0: cfg.sclk_polarity = false; cfg.sclk_phase = false; break;
6454:             case 1: cfg.sclk_polarity = false; cfg.sclk_phase = true; break;
6455:             case 2: cfg.sclk_polarity = true; cfg.sclk_phase = false; break;
6456:             case 3: cfg.sclk_polarity = true; cfg.sclk_phase = true; break;
6457:         }
6458:         error_code = SPI_Configuration_Set( ID_IDI48, &cfg );
6459:         if ( SUCCESS != error_code ) return error_code;
6460:         printf( "OK\n" );
6461:     }
6462:     return SUCCESS;
6463: }
6464:
6465: static int CMD__SPI_Config_SDI_Polarity( int board_id, int argc, char * argv[] )
6466: {
6467:     int error_code;
6468:
6469:     struct spi_cfg cfg;
6470:
6471:     error_code = SPI_Configuration_Get( board_id, &cfg );
6472:     if ( SUCCESS != error_code ) return error_code;
6473:
6474:     if ( argc < 1 )
6475:     {
6476:         printf( "SPI SDI POLARITY: %s\n", cfg.sdi_polarity ? "true" : "false" );
6477:     }
6478:     else
6479:     {
6480:         cfg.sdi_polarity = String_To_Bool( argv[0] );
6481:     }
6482:
```

```
6503:     error_code = SPI_Configuration_Set( board_id, &cfg );
6504:     if ( SUCCESS != error_code ) return error_code;
6505:     printf( "OK\n" );
6506: }
6507: return SUCCESS;
6508: }
6509:
6510: static int CMD__SPI_Config_SDO_Polarity( int board_id, int argc, char * argv[] )
6511: {
6512:     int error_code;
6513:
6514:     struct spi_cfg cfg;
6515:
6516:     error_code = SPI_Configuration_Get( board_id, &cfg );
6517:     if ( SUCCESS != error_code ) return error_code;
6518:
6519:     if ( argc < 1 )
6520:     {
6521:         printf( "SPI SDO POLARITY: %s\n", cfg.sdo_polarity ? "true" : "false" );
6522:     }
6523:     else
6524:     {
6525:         cfg.sdo_polarity = String_To_Bool( argv[0] );
6526:     }
6527:     error_code = SPI_Configuration_Set( board_id, &cfg );
6528:     if ( SUCCESS != error_code ) return error_code;
6529:     printf( "OK\n" );
6530: }
6531: return SUCCESS;
6532: }
6533:
6534: static int CMD__SPI_Config_SDIO_Wrap( int board_id, int argc, char * argv[] )
6535: {
6536:     int error_code;
6537:
6538:     struct spi_cfg cfg;
6539:
6540:     error_code = SPI_Configuration_Get( board_id, &cfg );
6541:     if ( SUCCESS != error_code ) return error_code;
6542:
6543:     if ( argc < 1 )
6544:     {
6545:         printf( "SPI wrap: %s\n", cfg.sdio_wrap ? "true" : "false" );
6546:     }
6547:     else
6548:     {
6549:         cfg.sdio_wrap = String_To_Bool( argv[0] );
6550:     }
6551:     error_code = SPI_Configuration_Set( board_id, &cfg );
6552:     if ( SUCCESS != error_code ) return error_code;
6553:     printf( "OK\n" );
6554: }
6555: return SUCCESS;
6556: }
6557:
6558: static int CMD__SPI_Config_Chip_Select_Route( int board_id, int argc, char * argv[] )
6559: {
6560:     int error_code;
6561:
6562:     struct spi_cfg cfg;
6563:
6564:     error_code = SPI_Configuration_Set( board_id, &cfg );
6565:     if ( SUCCESS != error_code ) return error_code;
6566:     printf( "OK\n" );
6567: }
6568: return SUCCESS;
6569: }
6570:
6571: static int CMD__SPI_Config_Clock_Polarity( int board_id, int argc, char * argv[] )
6572: {
6573:     int error_code;
6574:
6575:     struct spi_cfg cfg;
6576:
6577:     error_code = SPI_Configuration_Set( board_id, &cfg );
6578:     if ( SUCCESS != error_code ) return error_code;
6579:     printf( "OK\n" );
6580: }
```

```

6582: error_code = SPI_Configuration_Get( board_id, &cfg );
6583: if ( SUCCESS != error_code ) return error_code;
6584:
6585: if ( argc < 1 )
6586: {
6587:     printf( "SPI CS ROUTE: %s\n", cfg.chip_select_route_ch1 ? "1" : "0" );
6588: }
6589: else
6590: {
6591:     cfg.chip_select_route_ch1 = String_To_Bool( argv[0] );
6592:
6593:     error_code = SPI_Configuration_Set( board_id, &cfg );
6594:     if ( SUCCESS != error_code ) return error_code;
6595:     printf( "OK\n" );
6596: }
6597: return SUCCESS;
6598: }
6599:
6605: static int CMD__SPI_Config_Chip_Select_Behavior( int board_id, int argc, char * argv[] )
6606: {
6607:     int error_code;
6608:
6609:     struct spi_cfg cfg;
6610:
6611:
6612:     error_code = SPI_Configuration_Get( board_id, &cfg );
6613:     if ( SUCCESS != error_code ) return error_code;
6614:
6615:     if ( argc < 1 )
6616:     {
6617:         printf( "SPI CSB: " );
6618:         switch ( cfg.chip_select_behavior )
6619:         {
6620:             case CSB_SOFTWARE: printf( "software" ); break;
6621:             case CSB_BUFFER: printf( "buffer" ); break;
6622:             case CSB_uint8_t: printf( "uint8_t" ); break;
6623:             case CSB_uint16_t: printf( "uint16_t" ); break;
6624:             default: printf( "undefined" ); break;
6625:         }
6626:         printf( "\n" );
6627:     }
6628:     else
6629:     {
6630:         if ( 0 == strcmpi( "software", argv[0] ) ) cfg.chip_select_behavior = 0;
6631:         else if ( 0 == strcmpi( "buffer", argv[0] ) ) cfg.chip_select_behavior = 1;
6632:         else if ( 0 == strcmpi( "uint8_t", argv[0] ) ) cfg.chip_select_behavior = 2;
6633:         else if ( 0 == strcmpi( "uint16_t", argv[0] ) ) cfg.chip_select_behavior = 3;
6634:         else
6635:         {
6636:             cfg.chip_select_behavior = (SPI_CS_B_ENUM) strtol( argv[0], NULL, 0 );
6637:             switch ( cfg.chip_select_behavior )
6638:             {
6639:                 case CSB_SOFTWARE:
6640:                 case CSB_BUFFER:
6641:                 case CSB_uint8_t:
6642:                 case CSB_uint16_t:
6643:                     break;
6644:                 default:
6645:                     error_code = -EC_SPI_CS_B_OUT_OF_RANGE;
6646:                     break;
6647:             }
6648:         }
6649:         if ( SUCCESS != error_code ) return error_code;
6650:

```

```
6651:     error_code = SPI_Configuration_Set( board_id, &cfg );
6652:     if ( SUCCESS != error_code ) return error_code;
6653:     printf( "OK\n" );
6654: }
6655: return SUCCESS;
6656: }
6657:
6663: static int CMD__SPI_Status( int board_id, int argc, char * argv[] )
6664: {
6665:     int     error_code;
6666:     int     index;
6667:     struct spi_status    status;
6668:
6669:     if ( argc < 1 )
6670:     {
6671:         error_code = SPI_Status_Read( board_id, &status );
6672:         if ( SUCCESS != error_code ) return error_code;
6673:         SPI_Report_Status_Text( &status, stdout );
6674:     }
6675:     else
6676:     {
6677:         for ( index = 0; index < argc; index++ )
6678:         {
6679:             if ( 0 == strcmpi( "rx", argv[index] ) )
6680:             {
6681:                 error_code = SPI_Status_Read( board_id, &status );
6682:                 if ( SUCCESS != error_code ) return error_code;
6683:                 SPI_Report_Status_Text( &status, stdout );
6684:             }
6685:             else if ( 0 == strcmpi( "tx", argv[index] ) )
6686:             {
6687:                 error_code = SPI_Status_Write( board_id, &status );
6688:                 if ( SUCCESS != error_code ) return error_code;
6689:                 SPI_Report_Status_Text( &status, stdout );
6690:             }
6691:         }
6692:     }
6693:     return SUCCESS;
6694: }
6695:
6701: static int CMD__SPI_Data_Interpreter( int    arg_start,
6702:                                         int    argc,
6703:                                         char * argv[],
6704:                                         size_t tx_size,
6705:                                         size_t * tx_count,
6706:                                         uint8_t * tx_buffer
6707:                                         )
6708: {
6709:     size_t arg_qty;
6710:     size_t count;
6711:     int    index_arg;
6712:     int    index_byte;
6713:     uint8_t data_temp;
6714:     char * endptr;
6715:     char * sp;
6716:
6717:     arg_qty = (size_t)( argc - arg_start );
6718:
6719:     sp      = NULL;
6720:     index_arg = arg_start;
6721:     index_byte = 0;
6722:     count   = arg_qty;
6723:     while ( ( count != 0 ) && ( index_byte < tx_size ) )
6724:     {
```

```
6725:     data_temp = (uint8_t) strtol( argv[index_arg], &endptr, 0 );
6726:     if ( endptr == argv[index_arg] )
6727:     {
6728:         sp = &(argv[index_arg][0]);
6729:         while ( ( '\0' != *sp ) && ( index_byte < tx_size ) )
6730:         {
6731:             tx_buffer[index_byte] = (uint8_t)( *sp );
6732:             sp++; index_byte++;
6733:         }
6734:         if ( '\0' == *sp ) sp = NULL;
6735:     }
6736:     else
6737:     {
6738:         tx_buffer[index_byte] = data_temp;
6739:         index_byte++;
6740:         sp = NULL;
6741:     }
6742:     count--; index_arg++;
6743: }
6744:
6745: if ( count > 0 )
6746: {
6747: #if defined( __MSDOS__ )
6748:     printf( "Warning: ignored %d arguments", count );
6749: #else
6750:     printf( "Warning: ignored %lu arguments", count );
6751: #endif
6752:     if ( NULL != sp )
6753:     {
6754:         printf( ", and portion of string argument" );
6755:     }
6756:     printf( "\n" );
6757: }
6758:
6759: *tx_count = index_byte;
6760: return SUCCESS;
6761: }
6762:
6773: static int CMD__SPI_Data( int board_id, int argc, char * argv[] )
6774: {
6775:     int error_code;
6776:     size_t index;
6777:     size_t transfer_count;
6778:     size_t lines;
6779:     uint8_t * bp;
6780:
6781:     struct board_dataset * dataset;
6782:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
6783:
6784:
6785:
6786:     if ( argc < 1 ) return -EC_PARAMETER;
6787:
6788:     CMD__SPI_Data_Interpreter( 0,
6789:                                argc,
6790:                                argv,
6791:                                SPI_BLOCK_SIZE,
6792:                                &transfer_count,
6793:                                dataset->spi.tx_buffer
6794:                                );
6795:
6796:     error_code = SPI_Data_Write_Read( board_id,
6797:                                         sizeof( uint8_t ),
6798:                                         transfer_count,
```

```
6799:         dataset->spi.tx_buffer,
6800:         transfer_count,
6801:         dataset->spi.rx_buffer
6802:     );
6803:     if ( SUCCESS != error_code ) return error_code;
6804:
6805:     lines = transfer_count / HEX_DUMP_BYTES_PER_LINE;
6806:
6807:     for ( index = 0; index <= lines; index++ )
6808:     {
6809:         bp = &(dataset->spi.rx_buffer[index * HEX_DUMP_BYTES_PER_LINE]);
6810:         if ( transfer_count < HEX_DUMP_BYTES_PER_LINE )
6811:         {
6812:             Hex_Dump_Line( 0, transfer_count, bp, stdout );
6813:         }
6814:         else
6815:         {
6816:             Hex_Dump_Line( 0, HEX_DUMP_BYTES_PER_LINE, bp, stdout );
6817:             transfer_count = transfer_count - HEX_DUMP_BYTES_PER_LINE;
6818:         }
6819:     }
6820:     return SUCCESS;
6821: }
6822:
6823: static int CMD__SPI_FIFO( int board_id, int argc, char * argv[] )
6824: {
6825:     int      error_code;
6826:     int      index;
6827:     size_t   count;
6828:     SPI_CSBOARD_ENUM csb;
6829:     int      read_count;
6830:
6831:     uint8_t   tx_buffer[SPI_FIFO_SIZE];
6832:     uint8_t   rx_buffer[SPI_FIFO_SIZE];
6833:
6834:     struct board_dataset * dataset;
6835:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
6836:
6837:     read_count = 0;
6838:     if ( argc < 1 )
6839:     {
6840:         read_count = SPI_FIFO_SIZE;
6841:     }
6842:     else
6843:     {
6844:         if ( 0 == strcmpi( "rx", argv[0] ) )
6845:         {
6846:             if ( argc > 1 )
6847:             {
6848:                 if ( 0 == strcmpi( "all", argv[1] ) ) read_count = SPI_FIFO_SIZE;
6849:                 else read_count = (int) strtol( argv[1], NULL, 0 );
6850:             }
6851:         }
6852:         else if ( 0 == strcmpi( "tx", argv[0] ) )
6853:         {
6854:             CMD__SPI_Data_Interpreter( 1,
6855:                                         argc,
6856:                                         argv,
6857:                                         SPI_FIFO_SIZE,
6858:                                         &count,
6859:                                         tx_buffer
6860:                                     );
6861:         }
6862:     }
6863:     error_code = SPI_FIFO_Write( board_id, (void *) tx_buffer, sizeof( uint8_t ), count,
```

```

NULL );
6868:     if ( error_code < 0 ) return error_code;
6869:
6870:     printf( "OK, wrote %d objects or %d bytes\n", error_code, error_code * ((int)
6871:         sizeof( uint8_t )) );
6872:     return SUCCESS;
6873: }
6874: else if ( 0 == strcmpi( "commit", argv[0] ) )
6875: {
6876:     if ( argc > 1 )
6877:     {
6878:         if ( String_To_Bool( argv[1] ) ) SPI_Commit( board_id, 0xFF );
6879:         else
6880:             SPI_Commit( board_id, 0x00 );
6881:     }
6882: }
6883: else
6884: {
6885:     uint8_t commit_status;
6886:     SPI_Commit_Get( board_id, &commit_status );
6887:     printf( "commit status = %s\n", ( 0 == commit_status ) ? "false" : "true" );
6888:     return SUCCESS;
6889: }
6890:
6891: }
6892: }
6893:
6894:
6895:
6896:
6897: error_code = SPI_Configuration_Chip_Select_Behavior_Get( board_id, &csb );
6898: if ( SUCCESS != error_code ) return error_code;
6899:
6900: if ( SPI_Status_Write_FIFO_Is_Not_Empty( board_id ) )
6901: {
6902:     if ( CSB_SOFTWARE != csb ) SPI_Commit( board_id, 0xFF );
6903: }
6904:
6905: while ( SPI_Status_Write_FIFO_Is_Not_Empty( board_id ) )
6906: {
6907:     error_code = SUCCESS;
6908:     if ( dataset->quit_application ) error_code = -EC_APP_TERMINATE_CTRLC;
6909:     if ( Character_Get( NULL ) ) error_code = -EC_APP_TERMINATE;
6910:     if ( SUCCESS != error_code )
6911:     {
6912:         printf( "IDI_CMD__SPI_FIFO: waiting for tx empty\n" );
6913:         return -EC_APP_TERMINATE;
6914:     }
6915: }
6916:
6917: if ( read_count > 0 )
6918: {
6919:     int lines;
6920:     uint8_t * bp;
6921:
6922:     if ( SPI_Status_Read_FIFO_Is_Not_Empty( board_id ) )
6923:     {
6924:         error_code = SPI_FIFO_Read( board_id, (void *) rx_buffer, sizeof( uint8_t ),
6925:         read_count, NULL );
6926:         if ( error_code < 0 ) return error_code;
6927:         read_count = error_code * sizeof( uint8_t );
6928:         lines = read_count / HEX_DUMP_BYTES_PER_LINE;

```

```
6929:
6930:     for ( index = 0; index <= lines; index++ )
6931:     {
6932:         bp = &(rx_buffer[index * HEX_DUMP_BYTES_PER_LINE]);
6933:         if ( read_count < HEX_DUMP_BYTES_PER_LINE )
6934:         {
6935:             Hex_Dump_Line( 0, read_count, bp, stdout );
6936:         }
6937:     else
6938:     {
6939:         Hex_Dump_Line( 0, HEX_DUMP_BYTES_PER_LINE, bp, stdout );
6940:         read_count = read_count - HEX_DUMP_BYTES_PER_LINE;
6941:     }
6942: }
6943: }
6944: else
6945: {
6946:     printf( "FIFO Empty\n" );
6947: }
6948: }
6949: return SUCCESS;
6950:
6951:
6952:
6953:
6954: }
6955:
6961: static int CMD__SPI_Commit( int board_id, int argc, char * argv[] )
6962: {
6963:     int error_code;
6964:     uint8_t commit;
6965:
6966:     if ( argc < 1 )
6967:     {
6968:         uint8_t commit_status;
6969:         SPI_Commit_Get( board_id, &commit_status );
6970:         printf( "commit status = %s\n", ( 0 == commit_status ) ? "false" : "true" );
6971:     }
6972:     else if ( String_To_Bool( argv[0] ) ) commit = 0x01;
6973:     else
6974:         commit = 0x00;
6975:     error_code = SPI_Commit( board_id, commit );
6976:     if ( SUCCESS != error_code ) return error_code;
6977:
6978:     printf( "OK\n" );
6979:     return SUCCESS;
6980: }
6981:
6982:
6986: static struct command_line_board cmd_spi[] =
6987: {
6988:     { NULL, CMD__SPI_ID,           "id",    "wishbone id: params: none" },
6989:     { NULL, CMD__SPI_Config_Get,   "cfg",   "config dump: params: none" },
6990:     { NULL, CMD__SPI_Config_Clock_Hz, "clk",   "clk:           params: [<clock freq in hertz>]" },
6991:     { NULL, CMD__SPI_Config_End_Cycle_Delay_Sec, "ecd",   "end delay:   params: [<time in seconds>]" },
6992:     { NULL, CMD__SPI_Config_Mode,   "mode",  "mode:           params: [<0/1/2/3>]" },
6993:     { NULL, CMD__SPI_Config_SDIL_Polarity, "sdi",   "sdi pol:      params: [<true/false>]" },
6994:     { NULL, CMD__SPI_Config_SDO_Polarity, "sdo",   "sdo pol:      params: [<true/false>]" },
6995:     { NULL, CMD__SPI_Config_SDIO_Wrap,   "wrap",  "sdo-->sdi:    params: [<true/false>]" },
```

```

6996: { NULL, CMD__SPI_Config_Chip_Select_Behavior, "csb", "chip select behavior: params: [0/1/2/3/software/buffer/uint8_t/uint16_t]" },
6997: { NULL, CMD__SPI_Config_Chip_Select_Route, "csr", "chip select routing: params: [<true/1/false/0>] " },
6998: { NULL, CMD__SPI_Status, "status", "status of both TX and RX buffers" },
6999: { NULL, CMD__SPI_Data, "data", "read/write: params: <data list: bytes/characters/strings>" },
7000: { NULL, CMD__SPI_FIFO, "fifo", "fifo r/w: params: <data list: bytes/characters/strings>" },
7001: { NULL, CMD__SPI_Commit, "commit", "causes spi transactions to start" },
7002: { NULL, NULL, NULL, NULL }
7003: };
7004:
7010: int Command_Line_SPI( int board_id, int argc, char* argv[] )
7011: {
7012:     int error_code;
7013:     int index;
7014:     int argc_new;
7015:     char ** argv_new;
7016:
7017:     error_code = -EC_SYNTAX;
7018:
7019:     if ( argc < 1 ) return -EC_NOT_FOUND;
7020:
7021:     index = 0;
7022:     while ( NULL != cmd_spi[index].cmd_fnc )
7023:     {
7024:         if ( 0 == strcasecmp( cmd_spi[index].name, argv[0] ) )
7025:         {
7026:             argv_new = &(argv[1]);
7027:             argc_new = argc - 1;
7028:             if ( 0 == argc_new ) argv_new = NULL;
7029:
7030:         {
7031:             struct board_dataset * dataset;
7032:             dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
7033:             if ( IO_TRACE_START_SPI == dataset->trace.start )
7034:             {
7035:                 dataset->trace.active = true;
7036:             }
7037:         }
7038:
7039:         error_code = (* cmd_spi[index].cmd_fnc )( board_id, argc_new, argv_new );
7040:         break;
7041:     }
7042:     index++;
7043: }
7044: return error_code;
7045: }
7046:
7047:
7048:
7049:
7050:
7051:
7052:
7053:
7054:
7055:
7056:
7057: const double ADS1259_CLOCK_MHZ      = 7.3728e6;
7058: const double ADS1259_CLOCK_TOLERANCE = 0.02;
7059:
7060: const int32_t ADS1259_OFC_MAX = 8388607L;

```

```
7061: const int32_t ADS1259_OFC_MIN = -8388607L;
7062:
7063: const int32_t ADS1259_FSC_MAX = 8388608L;
7064: const int32_t ADS1259_FSC_MIN = 0L;
7065:
7066:
7067:
7068:
7071: enum
7072: { AS_ADS1259_CONFIG0 = 0,
7073: AS_ADS1259_CONFIG1 = 1,
7074: AS_ADS1259_CONFIG2 = 2,
7075: AS_ADS1259_OF0C0 = 3,
7076: AS_ADS1259_OF0C1 = 4,
7077: AS_ADS1259_OF0C2 = 5,
7078: AS_ADS1259_FSC0 = 6,
7079: AS_ADS1259_FSC1 = 7,
7080: AS_ADS1259_FSC2 = 8,
7081: AS_ADS1259_REG_QTY = 9
7082: };
7083:
7086: static char * as_register_name[ ] =
7087: {
7088: "config0",
7089: "config1",
7090: "config2",
7091: "ofc0",
7092: "ofc1",
7093: "ofc2",
7094: "fsc0",
7095: "fsc1",
7096: "fsc2",
7097: NULL
7098: };
7099:
7102: struct bit_string_info
7103: {
7104: size_t register_offset;
7105: size_t bit_offset;
7106: size_t bit_width;
7107: char * name;
7108: };
7109:
7112: static struct bit_string_info as_bit_string[] =
7113: {
7114: { AS_ADS1259_CONFIG0, 0, 1, "spi" },
7115: { AS_ADS1259_CONFIG0, 2, 1, "rbias" },
7116: { AS_ADS1259_CONFIG0, 4, 2, "id" },
7117: { AS_ADS1259_CONFIG1, 0, 3, "delay" },
7118: { AS_ADS1259_CONFIG1, 3, 1, "extref" },
7119: { AS_ADS1259_CONFIG1, 4, 1, "sinc2" },
7120: { AS_ADS1259_CONFIG1, 6, 1, "chksum" },
7121: { AS_ADS1259_CONFIG1, 7, 1, "flag" },
7122: { AS_ADS1259_CONFIG2, 0, 3, "dr" },
7123: { AS_ADS1259_CONFIG2, 4, 1, "pulse" },
7124: { AS_ADS1259_CONFIG2, 5, 1, "syncout" },
7125: { AS_ADS1259_CONFIG2, 6, 1, "extclk" },
7126: { AS_ADS1259_CONFIG2, 7, 1, "drdy_b" },
7127: { AS_ADS1259_REG_QTY, 0, 0, NULL }
7128: };
7129:
7132: static int ADS1259_Calibration_Range_Test( int32_t * value, size_t address, size_t
clamp_and_error_skip )
7133: {
```

```
7134: int32_t scratch;
7135: int error_code = SUCCESS;
7136:
7137: scratch = *value;
7138: if ( AS_AMS1259_OFC0 == address )
7139: {
7140:     if ( scratch > ADS1259_OFC_MAX )
7141:     {
7142:         error_code = -EC_RANGE;
7143:         scratch = ADS1259_OFC_MAX;
7144:     }
7145:     if ( scratch < ADS1259_OFC_MIN )
7146:     {
7147:         error_code = -EC_RANGE;
7148:         scratch = ADS1259_OFC_MIN;
7149:     }
7150: }
7151: else
7152: {
7153:     if ( scratch > ADS1259_FSC_MAX )
7154:     {
7155:         error_code = -EC_RANGE;
7156:         scratch = ADS1259_FSC_MAX;
7157:     }
7158:     if ( scratch < ADS1259_FSC_MIN )
7159:     {
7160:         error_code = -EC_RANGE;
7161:         scratch = ADS1259_FSC_MIN;
7162:     }
7163: }
7164:
7165: if ( 0 != clamp_and_error_skip )
7166: {
7167:     *value = scratch;
7168:     error_code = SUCCESS;
7169: }
7170: return error_code;
7171: }
7172:
7173: int AS_Register_Name_To_Offset( char * name )
7174: {
7175:     size_t index;
7176:
7177:     index = 0;
7178:     while ( NULL != as_register_name[index] )
7179:     {
7180:         if ( 0 == strcmpi( as_register_name[index], name ) )
7181:         {
7182:             return index;
7183:         }
7184:         index++;
7185:     }
7186:     return -EC_NOT_FOUND;
7187: }
7188: return -EC_NOT_FOUND;
7189: }
7190:
7191: int Bit_String_Report( struct bit_string_info * table, size_t address, uint8_t value,
FILE * out )
7192: {
7193:     size_t index;
7194:     BOOL not_done;
7195:     BOOL initial;
7196:     uint8_t bit_string_value;
7197:
7198:     initial = true;
```

```
7201: not_done = true;
7202: index = 0;
7203: while ( not_done )
7204: {
7205:     if ( NULL == table[index].name )
7206:     {
7207:         not_done = false;
7208:     }
7209:     else if ( address == table[index].register_offset )
7210:     {
7211:         size_t i;
7212:         uint8_t mask = 0;
7213:         for ( i = 0; i < table[index].bit_width; i++ ) mask = ( mask << 1 ) | 0x01;
7214:         bit_string_value = value >> table[index].bit_offset;
7215:         bit_string_value = bit_string_value & mask;
7216:         if ( initial )
7217:         {
7218:             fprintf( out, " : " ); initial = false;
7219:         }
7220:         fprintf( out, " %s=%d", table[index].name, (int) bit_string_value );
7221:     }
7222:     index++;
7223: }
7224: return SUCCESS;
7225: }
7226:
7227: int Bit_String_Index_By_Name( struct bit_string_info * table, char * name )
7228: {
7229:     int index;
7230:     index = 0;
7231:     while ( NULL != table[index].name )
7232:     {
7233:         if ( 0 == strcmp( table[index].name, name ) ) return index;
7234:         index++;
7235:     }
7236:     return -EC_NOT_FOUND;
7237: }
7238: return -EC_NOT_FOUND;
7239: }
7240:
7241: static int AS_Chip_Select_Route_Override( int board_id )
7242: {
7243:     struct board_dataset * dataset;
7244:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
7245:
7246:     #define FUNCTION_NAME__ "AS_Chip_Select_Route_Override"
7247:     #if ( IO_TRACE_USE_AS_LOGGING == 1 )
7248:         IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
7249:     #endif
7250:     #undef FUNCTION_NAME__
7251:
7252:     SPI_Configuration_Chip_Select_Route_Get( board_id, &(dataset->as_spi_cs_route_backup) );
7253:
7254:     SPI_Configuration_Chip_Select_Route_Set( board_id, dataset->as_cs_default );
7255:
7256:     return SUCCESS;
7257: }
7258:
7259: static int AS_Chip_Select_Route_Restore( int board_id )
7260: {
7261:     struct board_dataset * dataset;
7262:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
7263:
7264:     #define FUNCTION_NAME__ "AS_Chip_Select_Route_Restore"
7265:     #if ( IO_TRACE_USE_AS_LOGGING == 1 )
7266:         IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
7267:     #endif
7268:
```

```
7281: #undef FUNCTION_NAME__
7282:
7283: SPI_Configuration_Chip_Select_Route_Set( board_id, dataset->as_spi_cs_route_backup );
7284: return SUCCESS;
7285: }
7286:
7287: int AS_Opcode_Write( int board_id, uint8_t * tx_buf, size_t count )
7288: {
7289:     int error_code;
7290:
7291:
7292:
7293:
7294: #define FUNCTION_NAME__ "AS_Opcode_Write"
7295: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
7296:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
7297: #endif
7298: #undef FUNCTION_NAME__
7299:
7300:     AS_Chip_Select_Route_Override( board_id );
7301:     SPI_Configuration_Chip_Select_Behavior_Set( board_id, CSB_BUFFER );
7302:     error_code = SPI_Data_Write_Read( board_id,
7303:                                         1,
7304:                                         count,
7305:                                         tx_buf,
7306:                                         0,
7307:                                         NULL
7308:                                         );
7309:     AS_Chip_Select_Route_Restore( board_id );
7310:     return error_code;
7311: }
7312:
7313: static int CMD__AS_Wakeup( int board_id, int argc, char * argv[] )
7314: {
7315:     uint8_t tx_buf[1] = { 0x03 };
7316:     (void) argc;
7317:     (void) argv;
7318:
7319:
7320:
7321: #define FUNCTION_NAME__ "CMD__AS_Wakeup"
7322: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
7323:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
7324: #endif
7325: #undef FUNCTION_NAME__
7326:
7327:     return AS_Opcode_Write( board_id, tx_buf, 1 );
7328: }
7329:
7330: static int CMD__AS_Sleep( int board_id, int argc, char * argv[] )
7331: {
7332:     uint8_t tx_buf[1] = { 0x05 };
7333:     (void) argc;
7334:     (void) argv;
7335:
7336:
7337:
7338: #define FUNCTION_NAME__ "CMD__AS_Sleep"
7339: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
7340:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
7341: #endif
7342: #undef FUNCTION_NAME__
7343:
7344:     return AS_Opcode_Write( board_id, tx_buf, 1 );
7345: }
7346:
7347: static int CMD__AS_Reset( int board_id, int argc, char * argv[] )
7348: {
7349:     uint8_t tx_buf[1] = { 0x07 };
7350:     (void) argc;
```

```
7353: (void) argv;
7354:
7355: #define FUNCTION_NAME__ "CMD__AS_Reset"
7356: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
7357: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
7358: #endif
7359: #undef FUNCTION_NAME__
7360:
7361: return AS_Opcode_Write( board_id, tx_buf, 1 );
7362: }
7363:
7364: static int CMD__AS_Start( int board_id, int argc, char * argv[] )
7365: {
7366: uint8_t tx_buf[1] = { 0x09 };
7367: (void) argc;
7368: (void) argv;
7369:
7370: #define FUNCTION_NAME__ "CMD__AS_Start"
7371: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
7372: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
7373: #endif
7374: #undef FUNCTION_NAME__
7375:
7376: #define FUNCTION_NAME__ "CMD__AS_Stop"
7377: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
7378: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
7379: #endif
7380: #undef FUNCTION_NAME__
7381:
7382: static int CMD__AS_Stop( int board_id, int argc, char * argv[] )
7383: {
7384: uint8_t tx_buf[1] = { 0x0B };
7385: (void) argc;
7386: (void) argv;
7387:
7388: #define FUNCTION_NAME__ "CMD__AS_Stop"
7389: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
7390: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
7391: #endif
7392: #undef FUNCTION_NAME__
7393:
7394: #define FUNCTION_NAME__ "CMD__AS_Rdataac"
7395: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
7396: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
7397: #endif
7398: #undef FUNCTION_NAME__
7399:
7400: static int CMD__AS_Rdataac( int board_id, int argc, char * argv[] )
7401: {
7402: uint8_t tx_buf[1] = { 0x10 };
7403: (void) argc;
7404: (void) argv;
7405:
7406: #define FUNCTION_NAME__ "CMD__AS_Rdataac"
7407: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
7408: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
7409: #endif
7410: #undef FUNCTION_NAME__
7411:
7412: return AS_Opcode_Write( board_id, tx_buf, 1 );
7413: }
7414:
7415: static int CMD__AS_Sdataac( int board_id, int argc, char * argv[] )
7416: {
7417: uint8_t tx_buf[1] = { 0x11 };
7418: (void) argc;
7419: (void) argv;
7420:
7421: #define FUNCTION_NAME__ "CMD__AS_Sdataac"
7422: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
```

```
7425: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
7426: #endif
7427: #undef FUNCTION_NAME__
7428:
7429: return AS_Opcode_Write( board_id, tx_buf, 1 );
7430: }
7431:
7432: static int AS_Registers_Read( int board_id, size_t address, size_t count, uint8_t *
tx_buf, uint8_t * rx_buf )
7433: {
7434:     size_t object_count;
7435:     int error_code;
7436:     struct board_dataset * dataset;
7437:
7438:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
7439:     #define FUNCTION_NAME__ "AS_Registers_Read"
7440:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
7441:     #endif
7442:     #if ( IO_TRACE_USE_AS_LOGGING == 1 )
7443:         IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
7444:     #endif
7445:     #undef FUNCTION_NAME__
7446:
7447:
7448:     if ( ( address + count ) > AS_AM2315_REGISTER_QTY ) return
-EC_REGISTER_ADDRESS_EXCEEDED;
7449:
7450:     tx_buf[0] = 0x20 + ( (uint8_t) ( 0x0F & address ) );
7451:     if ( 0 == count )
7452:     {
7453:         tx_buf[1] = 0;
7454:         object_count = 2;
7455:         count = 1;
7456:     }
7457:     else
7458:     {
7459:         tx_buf[1] = ( (uint8_t) ( 0x0F & count ) ) - 1;
7460:         object_count = ( 0x0F & count ) + 2;
7461:     }
7462:
7463:     if ( true == dataset->set__suppress_io_activity )
7464:     {
7465:         size_t index;
7466:         for ( index = 0; index < count; index++ )
7467:         {
7468:             rx_buf[index + 2] = dataset->ads1259.reg[address + index];
7469:         }
7470:         return SUCCESS;
7471:     }
7472:
7473:
7474:     AS_Chip_Select_Route_Override( board_id );
7475:     SPI_Configuration_Chip_Select_Behavior_Set( board_id, CSB_BUFFER );
7476:     error_code = SPI_Data_Write_Read( board_id,
7477:         1,
7478:         object_count,
7479:         tx_buf,
7480:         object_count,
7481:         rx_buf
7482:     );
7483:     AS_Chip_Select_Route_Restore( board_id );
7484:     return error_code;
7485: }
7486:
7487: static int AS_Registers_Write( int board_id, size_t address, size_t count, uint8_t *
tx_buf, uint8_t * rx_buf )
```

```
7490: {
7491:     size_t      object_count;
7492:     size_t      index;
7493:     int         error_code;
7494:     struct board_dataset * dataset;
7495:
7496:     error_code = SUCCESS;
7497:     dataset    = ( struct board_dataset * ) board_definition[board_id].dataset;
7498:
7499: #define FUNCTION_NAME__ "AS_Registers_Write"
7500: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
7501:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
7502: #endif
7503: #undef FUNCTION_NAME__
7504:
7505:
7506:     if ( ( address + count ) > AS_AM2315_REGISTER_QTY ) return
-EC_REGISTER_ADDRESS_EXCEEDED;
7507:
7508:
7509:     tx_buf[0] = 0x40 + ( (uint8_t) ( 0x0F & address ) );
7510:     if ( 0 == count )
7511:     {
7512:         tx_buf[1]      = 0;
7513:         count        = 1;
7514:         object_count = 3;
7515:     }
7516:     else
7517:     {
7518:         tx_buf[1] = ( (uint8_t) ( 0x0F & count ) ) - 1;
7519:         object_count = ( 0x0F & count ) + 2;
7520:     }
7521:
7522:     if ( true == dataset->set__suppress_io_activity )
7523:     {
7524:
7525:         for ( index = 0; index < count; index++ )
7526:         {
7527:             dataset->ads1259.reg[index + address] = tx_buf[index + 2];
7528:         }
7529:         return SUCCESS;
7530:     }
7531:
7532:
7533:     AS_Chip_Select_Route_Override( board_id );
7534:     SPI_Configuration_Chip_Select_Behavior_Set( board_id, CSB_BUFFER );
7535:     error_code = SPI_Data_Write_Read( board_id,
7536:                                         1,
7537:                                         object_count,
7538:                                         tx_buf,
7539:                                         object_count,
7540:                                         rx_buf
7541:                                         );
7542:     AS_Chip_Select_Route_Restore( board_id );
7543:
7544:     return error_code;
7545: }
7546:
7547: static int CMD__AS_RReg( int board_id, int argc, char * argv[] )
7548: {
7549:     size_t      count;
7550:     size_t      index;
7551:     size_t      address;
7552:     int         error_code;
```

```

7555:    BOOL dump_as_hex;
7556:    uint8_t tx_buf[12] = { 0x00 };
7557:    uint8_t rx_buf[12] = { 0x00 };
7558:
7559: #define FUNCTION_NAME__ "CMD_AS_RReg"
7560: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
7561:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
7562: #endif
7563: #undef FUNCTION_NAME__
7564:
7565:    dump_as_hex = false;
7566:    index = 0;
7567:    if ( ( argc < 1 ) || ( NULL == argv ) )
7568:    {
7569:        count = AS_AM2315_REGISTER_QTY;
7570:        address = AS_AM2315_CONFIG0;
7571:        index = 1;
7572:    }
7573:    else
7574:    {
7575:        if ( ( 'a' == argv[0][0] ) || ( 'A' == argv[0][0] ) )
7576:        {
7577:            count = AS_AM2315_REGISTER_QTY;
7578:            address = AS_AM2315_CONFIG0;
7579:            index = 1;
7580:        }
7581:        else if ( 1 == argc )
7582:        {
7583:            count = 1;
7584:            error_code = AS_Register_Name_To_Offset( argv[0] );
7585:            if ( error_code < 0 )
7586:            {
7587:                address = (size_t) strtol( argv[0], NULL, 0 );
7588:            }
7589:            else
7590:            {
7591:                address = (size_t) error_code;
7592:            }
7593:            index = 1;
7594:        }
7595:    else
7596:    {
7597:
7598:        count = (size_t) strtol( argv[1], NULL, 0 );
7599:        error_code = AS_Register_Name_To_Offset( argv[0] );
7600:        if ( error_code < 0 )
7601:        {
7602:            address = (size_t) strtol( argv[0], NULL, 0 );
7603:        }
7604:        else
7605:        {
7606:            address = (size_t) error_code;
7607:        }
7608:        index = 2;
7609:    }
7610:
7611:    if ( argc > index )
7612:    {
7613:        if ( ( 'h' == argv[index][0] ) || ( 'H' == argv[index][0] ) )
7614:        {
7615:            dump_as_hex = true;
7616:        }
7617:    }
7618: }
```

```
7619:
7620:     error_code = AS_Registers_Read( board_id, address, count, tx_buf, rx_buf );
7621:     if ( error_code < 0 ) return error_code;
7622:
7623:     if ( dump_as_hex )
7624:     {
7625:         Hex_Dump_Line( (uint16_t) address, count, &(rx_buf[2]), stdout );
7626:     }
7627:     else
7628:     {
7629:         for ( index = 0; index < count; index++ )
7630:         {
7631:             printf( "%s=0x%02X", as_register_name[address + index], rx_buf[index + 2] );
7632:             Bit_String_Report( as_bit_string, address + index, rx_buf[index + 2], stdout );
7633:             printf( "\n" );
7634:         }
7635:     }
7636:     return SUCCESS;
7637: }
7638:
7641: enum buffer_column { BUFFER_COLUMN_VALUE = 0, BUFFER_COLUMN_TAG = 1 };
7642: #define BUFFER_COLUMN 2
7643: #define BUFFER_LENGTH 2048
7644: static int32_t buffer[BUFFER_COLUMN][BUFFER_LENGTH];
7645: static size_t buffer_index;
7646:
7649: static void Buffer_Init(void)
7650: {
7651:     size_t index;
7652:     buffer_index = 0;
7653:     for ( index = 0; index < BUFFER_LENGTH; index++ )
7654:     {
7655:         buffer[BUFFER_COLUMN_VALUE][index] = 0;
7656:         buffer[BUFFER_COLUMN_TAG][index] = 0;
7657:     }
7658: }
7659:
7662: static int Buffer_Mean_Int32( int32_t (* buffer)[BUFFER_LENGTH], size_t length, double
* mean )
7663: {
7664:     size_t index;
7665:     double sum;
7666:     int32_t * buf;
7667:
7668:     if ( 0 == length ) return -EC_PARAMETER;
7669:     sum = 0.0;
7670:     buf = buffer[BUFFER_COLUMN_VALUE];
7671:     for ( index = 0; index < length; index++ )
7672:     {
7673:         sum += (double) buf[index];
7674:     }
7675:     *mean = sum / ((double)length);
7676:     return SUCCESS;
7677: }
7678:
7681: static int Buffer_Standard_Deviation_Int32( int32_t (* buffer)[BUFFER_LENGTH], size_t
length, double mean, double * stdev )
7682: {
7683:     size_t index;
7684:     double term;
7685:     double squared_sum;
7686:     int32_t * buf;
7687:
7688:     if ( length < 2 ) return -EC_PARAMETER;
```

```
7689:  
7690:     squared_sum = 0.0;  
7691:     buf = buffer[BUFFER_COLUMN_VALUE];  
7692:     for ( index = 0; index < length; index++ )  
7693:     {  
7694:         term = ((double) buf[index]) - mean;  
7695:         squared_sum += term * term;  
7696:     }  
7697:     squared_sum = squared_sum / ((double) ( length - 1));  
7698:     *stdev = sqrt( squared_sum );  
7699:     return SUCCESS;  
7700: }  
7701:  
7704: static int Buffer_Minimum_Int32( int32_t (* buffer)[BUFFER_LENGTH], size_t length,  
double * minimum )  
7705: {  
7706:     int32_t min;  
7707:     size_t index;  
7708:     int32_t * buf;  
7709:  
7710:     if ( 0 == length ) return -EC_PARAMETER;  
7711:     min = INT32_MAX;  
7712:     buf = buffer[BUFFER_COLUMN_VALUE];  
7713:     for ( index = 0; index < length; index++ )  
7714:     {  
7715:         if ( buf[index] < min ) min = buf[index];  
7716:     }  
7717:     *minimum = (double) min;  
7718:     return SUCCESS;  
7719: }  
7720:  
7723: static int Buffer_Maximum_Int32( int32_t (* buffer)[BUFFER_LENGTH], size_t length,  
double * maximum )  
7724: {  
7725:     int32_t max;  
7726:     size_t index;  
7727:     int32_t * buf;  
7728:  
7729:     if ( 0 == length ) return -EC_PARAMETER;  
7730:     max = INT32_MIN;  
7731:     buf = buffer[BUFFER_COLUMN_VALUE];  
7732:     for ( index = 0; index < length; index++ )  
7733:     {  
7734:         if ( buf[index] > max ) max = buf[index];  
7735:     }  
7736:     *maximum = (double) max;  
7737:     return SUCCESS;  
7738: }  
7739:  
7742: static int Buffer_Peak_To_Peak_Int32( double min, double max, double * peak_to_peak )  
7743: {  
7744:     *peak_to_peak = max - min;  
7745:     return SUCCESS;  
7746: }  
7747:  
7750: static int Buffer_Stuff( int32_t value, int32_t tag )  
7751: {  
7752:     if ( buffer_index >= BUFFER_LENGTH ) return -EC_BUFFER_EXCEEDED;  
7753:     buffer[BUFFER_COLUMN_VALUE][buffer_index] = value;  
7754:     buffer[BUFFER_COLUMN_TAG][buffer_index] = tag;  
7755:     buffer_index++;  
7756:     return SUCCESS;  
7757: }  
7758:
```

```
7761: static int Buffer_Length( size_t * length )
7762: {
7763:     *length = buffer_index;
7764:     return SUCCESS;
7765: }
7766:
7769: static int Buffer_Save( FILE * out )
7770: {
7771:     size_t index;
7772:     size_t column;
7773:
7774:     for ( index = 0; index < buffer_index; index++ )
7775:     {
7776:         for ( column = 0; column < BUFFER_COLUMN; column++ )
7777:         {
7778:             #if defined( __MSDOS__ )
7779:                 fprintf( out, "%ld", buffer[column][index] );
7780:             #else
7781:                 fprintf( out, "%d",   buffer[column][index] );
7782:             #endif
7783:             if ( column < (BUFFER_COLUMN - 1) ) fprintf( out, ", " );
7784:             else                                fprintf( out, "\n" );
7785:         }
7786:     }
7787:     return SUCCESS;
7788: }
7789:
7790:
7793: static int Buffer_Save_Binary_Int32( const char * file_name )
7794: {
7795:     FILE * fd;
7796:     size_t index;
7797:     size_t column;
7798:
7799:     if ( 0 == buffer_index ) return -SUCCESS;
7800:
7801:     fd = fopen( file_name, "wb" );
7802:     if ( NULL == fd ) return -EC_FILE_ERROR;
7803:
7804:     for ( index = 0; index < buffer_index; index++ )
7805:     {
7806:         for ( column = 0; column < BUFFER_COLUMN; column++ )
7807:         {
7808:             fwrite( &(buffer[column][index]),
7809:                     sizeof( int32_t ),
7810:                     1,
7811:                     fd
7812:                 );
7813:         }
7814:     }
7815: #if(0)
7816:
7817:     fwrite( buffer,
7818:             BUFFER_COLUMN * sizeof( int32_t ),
7819:             buffer_index,
7820:             fd
7821:         );
7822: #endif
7823:     fclose( fd );
7824:
7825:     return SUCCESS;
7826: }
7827:
7828:
```

```

7831: static int AS_Rdata_Statistics( FILE * out )
7832: {
7833:     int error_code;
7834:     size_t length;
7835:     double mean;
7836:     double maximum;
7837:     double minimum;
7838:     double peak_to_peak;
7839:     double stdev;
7840:
7841:     Buffer_Length( &length );
7842:
7843:     Buffer_Mean_Int32( buffer, length, &mean );
7844:     Buffer_Minimum_Int32( buffer, length, &minimum );
7845:     Buffer_Maximum_Int32( buffer, length, &maximum );
7846:     Buffer_Peak_To_Peak_Int32( minimum, maximum, &peak_to_peak );
7847:     error_code = Buffer_Standard_Deviation_Int32( buffer, length, mean, &stdev );
7848:     fprintf( out, "Data Statistics:\n" );
7849:     fprintf( out, "    sample_quantity      = %7d\n", (int) length );
7850:     fprintf( out, "    mean                  = %7.3f\n", mean );
7851:     if ( SUCCESS == error_code )
7852:     {
7853:         fprintf( out, "    stdev                 = %7.3f\n", stdev );
7854:     }
7855:     fprintf( out, "    minimum                = %7.3f\n", minimum );
7856:     fprintf( out, "    maximum                = %7.3f\n", maximum );
7857:     fprintf( out, "    peak-to-peak          = %7.3f\n", peak_to_peak );
7858:
7859:     fprintf( out, "\n" );
7860:     return SUCCESS;
7861: }
7862:
7863: static int AS_Data_Ready_NoWait( int board_id, size_t drdy_anticipated_active )
7864: {
7865:     uint8_t tx_buf[4] = { 0x00, 0x00, 0x00, 0x00 };
7866:     uint8_t rx_buf[4] = { 0x00, 0x00, 0x00, 0x00 };
7867:     uint8_t scratch;
7868:
7869:     if ( 0 == drdy_anticipated_active ) scratch = 0x00;
7870:     else
7871:         scratch = 0x80;
7872:
7873:     rx_buf[2] = scratch;
7874:     AS_Registers_Read( board_id, 2, 1, tx_buf, rx_buf );
7875:
7876:     if ( scratch == ( rx_buf[2] & 0x80 ) ) return 0;
7877:
7878:     return 1;
7879:
7880: }
7881:
7882:
7883:
7884: static void AS_Data_Ready_Wait( int board_id, size_t drdy_anticipated_active, size_t *
7885: loop_count_per_data_ready )
7886: {
7887:     size_t test_count;
7888:     uint8_t tx_buf[4] = { 0x00, 0x00, 0x00, 0x00 };
7889:     uint8_t rx_buf[4] = { 0x00, 0x00, 0x00, 0x00 };
7890:     uint8_t scratch;
7891:
7892:     if ( 0 == drdy_anticipated_active ) scratch = 0x00;
7893:     else
7894:         scratch = 0x80;
7895:
7896:     test_count = 0;
7897:     rx_buf[2] = scratch;
7898:     while ( scratch == ( rx_buf[2] & 0x80 ) )
7899:
7900:
```

```
7901: {
7902:     AS_Registers_Read( board_id, 2, 1, tx_buf, rx_buf );
7903:     test_count++;
7904: }
7905: if ( test_count > 1 )
7906: {
7907:     if ( NULL != loop_count_per_data_ready )
7908:     {
7909:         *loop_count_per_data_ready = 0;
7910:     }
7911: }
7912: }
7913:
7914: static int AS_Data_Read( int board_id, int32_t * adc_value )
7915: {
7916:     int error_code;
7917:     uint32_t adc_temp;
7918:     size_t byte_index;
7919:     uint8_t tx_buf[4] = { 0x12, 0x00, 0x00, 0x00 };
7920:     uint8_t rx_buf[4] = { 0x00, 0x00, 0x00, 0x00 };
7921:
7922:
7923:
7924:
7925:
7926:     AS_Chip_Select_Route_Override( board_id );
7927:     SPI_Configuration_Chip_Select_Behavior_Set( board_id, CSB_BUFFER );
7928:     error_code = SPI_Data_Write_Read( board_id,
7929:                                         1,
7930:                                         4,
7931:                                         tx_buf,
7932:                                         4,
7933:                                         rx_buf
7934:                                         );
7935:     AS_Chip_Select_Route_Restore( board_id );
7936:     if ( error_code < 0 ) goto AS_Read_Data_Exit;
7937:
7938:     adc_temp = 0;
7939:     for ( byte_index = 1; byte_index < 4; byte_index++ ) adc_temp = (adc_temp << 8) + (
7940: (uint32_t) rx_buf[byte_index] );
7941:
7942:     if ( 0 != ( 0xFF800000L & adc_temp ) )
7943:     {
7944:         adc_temp = 0xFF800000L | adc_temp;
7945:     }
7946:     *adc_value = (int32_t) adc_temp;
7947:
7948: AS_Read_Data_Exit:
7949:     return error_code;
7950: }
7951:
7952:
7953:
7954:
7955: static void AS_Time_Statistics( STOPWATCH_HANDLE_TYPE * stopwatch_handle,
7956:                               int time_overall_index,
7957:                               int time_reading_index,
7958:                               int time_writing_index,
7959:                               FILE * out
7960:                               )
7961:
7962:
7963: {
7964:     STOPWATCH_TIMEVAL_TYPE * time_value;
7965:     double time_result[STOPWATCH_QUANTITY];
7966:     size_t sample_count;
7967:     double dscratch;
7968:
```

```

7969: Buffer_Length( &sample_count );
7970:
7971: fprintf( out, "Timing Statistics:\n" );
7972: time_value = StopWatch_Value( stopwatch_handle[time_overall_index] );
7973: time_result[time_overall_index] = ( (double) time_value->tv_sec ) + ( (double)
time_value->tv_usec ) * 0.000001;
7974: fprintf( out, " Total time           = %7.3f sec, 100%%\n",
time_result[time_overall_index] );
7975:
7976: if ( 0.0 == time_result[time_overall_index] ) return;
7977:
7978: time_value = StopWatch_Value( stopwatch_handle[time_reading_index] );
7979: time_result[time_reading_index] = ( (double) time_value->tv_sec ) + ( (double)
time_value->tv_usec ) * 0.000001;
7980: fprintf( out, " Read timing          = %7.3f sec, %5.1f%%\n",
time_result[time_reading_index], ( time_result[time_reading_index]* 100.0 /
time_result[time_overall_index] ) );
7981:
7982: time_value = StopWatch_Value( stopwatch_handle[time_writing_index] );
7983: time_result[time_writing_index] = ( (double) time_value->tv_sec ) + ( (double)
time_value->tv_usec ) * 0.000001;
7984: fprintf( out, " To File(s) timing   = %7.3f sec, %5.1f%%\n",
time_result[time_writing_index], ( time_result[time_writing_index]* 100.0 /
time_result[time_overall_index] ) );
7985:
7986: fprintf( out, " Total Samples        = %7d\n", (int) sample_count );
7987: dscratch = ( (double) sample_count ) / time_result[time_overall_index];
7988: fprintf( out, " Samples/Sec          = %7.2f Samples/Sec\n", dscratch );
7989: dscratch = 1000000.0 / dscratch;
7990: fprintf( out, " Time per sample     = %7.3f uSec/Sample\n", dscratch );
7991: }
7992:
7993:
7994:
8007: static int CMD_AS_Rdata( int board_id, int argc, char * argv[] )
8008: {
8009: char baton[4] = { '/', '-', '\\', '|' };
8010: const size_t baton_max = 3;
8011: size_t baton_index;
8012: int32_t adc_value;
8013:
8014:
8015: size_t index;
8016: size_t loop_count_per_data_ready;
8017: size_t sample_qty;
8018: FILE * fd_text;
8019: char * file_name_text;
8020: char * file_name_binary;
8021: size_t file_request;
8022: size_t file_valid;
8023: BOOL verbose;
8024: BOOL use_environment_ads1259_registers;
8025: int error_code;
8026: struct board_dataset * dataset;
8027: const size_t FILE_TYPE_TEXT = 0x01;
8028: const size_t FILE_TYPE_BINARY = 0x02;
8029:
8030:
8031:
8032: STOPWATCH_HANDLE_TYPE stopwatch_handle[STOPWATCH_QUANTITY];
8033: enum { TIME_OVERALL = 0, TIME_READING = 1, TIME_WRITING = 2 } ;
8034:
8035: (void) argc;
8036: (void) argv;

```

```
8037:
8038: dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
8039:
8040: #define FUNCTION_NAME__ "CMD_AS_Rdata"
8041: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
8042: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
8043: #endif
8044: #undef FUNCTION_NAME__
8045:
8046: fd_text      = NULL;
8047: file_request = 0;
8048: file_valid   = 0;
8049: index        = 0;
8050: sample_qty   = 1;
8051: verbose      = false;
8052: use_environment_ads1259_registers = false;
8053:
8054: while ( index < argc )
8055: {
8056:     if ( ( FILE_TYPE_TEXT | FILE_TYPE_BINARY ) & file_request )
8057:     {
8058:         if ( FILE_TYPE_TEXT & file_request )
8059:         {
8060:             file_name_text = argv[index];
8061:             file_request = file_request & ~FILE_TYPE_TEXT;
8062:             file_valid   = file_valid | FILE_TYPE_TEXT;
8063:
8064:             fd_text = fopen( file_name_text, "wt" );
8065:         }
8066:         if ( FILE_TYPE_BINARY & file_request )
8067:         {
8068:             file_name_binary = argv[index];
8069:             file_request = file_request & ~FILE_TYPE_BINARY;
8070:             file_valid   = file_valid | FILE_TYPE_BINARY;
8071:         }
8072:     }
8073:     else if ( 0 == strcmpi( "--text", argv[index] ) )
8074:     {
8075:         file_request = file_request | FILE_TYPE_TEXT;
8076:     }
8077:     else if ( 0 == strcmpi( "--binary", argv[index] ) )
8078:     {
8079:         file_request = file_request | FILE_TYPE_BINARY;
8080:     }
8081:     else if ( 0 == strcmpi( "--v", argv[index] ) )
8082:     {
8083:         verbose = true;
8084:     }
8085:     else if ( 0 == strcmpi( "--e", argv[index] ) )
8086:     {
8087:         use_environment_ads1259_registers = true;
8088:     }
8089:     else
8090:     {
8091:         sample_qty = (size_t) strtol( argv[index], NULL, 0 );
8092:         if ( sample_qty > ( BUFFER_LENGTH - 1 ) ) sample_qty = BUFFER_LENGTH;
8093:     }
8094:     index++;
8095: }
8096:
8097: CMD_AS_Reset( board_id, 0, NULL );
8098: Buffer_Init();
8099:
8100:
```

```
8101: StopWatch_Initialization();
8102:
8103: stopwatch_handle[TIME_OVERALL] = StopWatch_Open();
8104: StopWatch_Set( stopwatch_handle[TIME_OVERALL], STOPWATCH_RESET );
8105: stopwatch_handle[TIME_READING] = StopWatch_Open();
8106: StopWatch_Set( stopwatch_handle[TIME_READING], STOPWATCH_RESET );
8107: stopwatch_handle[TIME_WRITING] = StopWatch_Open();
8108: StopWatch_Set( stopwatch_handle[TIME_WRITING], STOPWATCH_RESET );
8109:
8110:
8111:
8112: #if defined( __MSDOS__ )
8113: {
8114:     double scratch;
8115:
8116:     scratch = ADS1259_CLOCK_MHZ * ( 1.0 + ADS1259_CLOCK_TOLERANCE );
8117:     scratch = 65536.0 / scratch;
8118:     scratch = scratch * 1000.0;
8119:     scratch = scratch + 100.0;
8120:     delay ( (size_t) scratch );
8121: }
8122: #endif
8123: CMD__AS_Sdatac( board_id, 0, NULL );
8124: CMD__AS_Stop( board_id, 0, NULL );
8125:
8126:
8127:
8128:
8129: loop_count_per_data_ready = 0;
8130:
8131: if ( use_environment_ads1259_registers )
8132: {
8133:     uint8_t tx_buf[12] = { 0x00 };
8134:     uint8_t rx_buf[12] = { 0x00 };
8135:
8136:     for ( index = 0; index < AS_ADS1259_REGISTER_QTY; index ++ )
8137:     {
8138:         tx_buf[index + 2] = dataset->ads1259.reg[index];
8139:     }
8140:
8141:     AS_Registers_Write( board_id, AS_ADS1259_CONFIG0, AS_ADS1259_REGISTER_QTY, tx_buf,
rx_buf );
8142: }
8143:
8144: if ( true == verbose )
8145: {
8146:     CMD__AS_RReg( board_id, 0, NULL );
8147:     baton_index = 0;
8148:     printf( "\nActive: " );
8149: }
8150:
8151: StopWatch_Set( stopwatch_handle[TIME_OVERALL], STOPWATCH_START );
8152:
8153:
8154:
8155:
8156:
8157: error_code = AS_Data_Read( board_id, &adc_value );
8158:
8159:
8160: for ( index = 0; index < sample_qty; index++ )
8161: {
8162:
8163:
```

```
8164: StopWatch_Set( stopwatch_handle[TIME_READING], STOPWATCH_START );
8165:
8166: CMD__AS_Start( board_id, 0, NULL );
8167: loop_count_per_data_ready++;
8168: AS_Data_Ready_Wait( board_id,
8169:     1,
8170:     &loop_count_per_data_ready
8171: );
8172:
8173: error_code = AS_Data_Read( board_id, &adc_value );
8174: if ( error_code < 0 ) goto CMD__AS_Rdata_Error_Exit;
8175:
8176:
8177: StopWatch_Set( stopwatch_handle[TIME_READING], STOPWATCH_PAUSE );
8178: StopWatch_Set( stopwatch_handle[TIME_WRITING], STOPWATCH_START );
8179:
8180: Buffer_Stuff( adc_value, loop_count_per_data_ready );
8181:
8182: StopWatch_Set( stopwatch_handle[TIME_WRITING], STOPWATCH_PAUSE );
8183:
8184: if ( true == verbose )
8185: {
8186:     printf( "\b%c", baton[baton_index] );
8187:     if ( baton_index == baton_max ) baton_index = 0;
8188:     else
8189:         baton_index++;
8190: }
8191:
8192: StopWatch_Set( stopwatch_handle[TIME_OVERALL], STOPWATCH_PAUSE );
8193:
8194: if ( true == verbose )
8195: {
8196:     printf( "\n" );
8197: }
8198:
8199: if ( FILE_TYPE_TEXT & file_valid )
8200: {
8201:     Buffer_Save( fd_text );
8202:     AS_Rdata_Statistics( fd_text );
8203:     AS_Time_Statistics( stopwatch_handle, TIME_OVERALL, TIME_READING, TIME_WRITING,
fd_text );
8204:     fclose( fd_text );
8205: }
8206: if ( FILE_TYPE_BINARY & file_valid )
8207: {
8208:     Buffer_Save_Binary_Int32( file_name_binary );
8209: }
8210:
8211:
8212: AS_Time_Statistics( stopwatch_handle, TIME_OVERALL, TIME_READING, TIME_WRITING, stdout
);
8213:
8214:
8215: AS_Rdata_Statistics( stdout );
8216:
8217: CMD__AS_Stop( board_id, 0, NULL );
8218:
8219: StopWatch_Termination();
8220:
8221: return SUCCESS;
8222: CMD__AS_Rdata_Error_Exit:
8223: AS_Chip_Select_Route_Restore( board_id );
8224: if ( FILE_TYPE_TEXT & file_valid ) fclose( fd_text );
8225:
```

```
8226: StopWatch_Termination();
8227:
8228: return error_code;
8229: }
8230:
8231: static int AS_Calibration_Value_Write( int board_id, size_t address, int32_t value )
8232: {
8233:     int error_code;
8234:     uint32_t scratch;
8235:     size_t index;
8236:     const size_t byte_length = 3;
8237:     uint8_t tx_buf[12] = { 0x00 };
8238:     uint8_t rx_buf[12] = { 0x00 };
8239:
8240:
8241:
8242: #define FUNCTION_NAME__ "AS_Calibration_Value_Write"
8243: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
8244:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
8245: #endif
8246: #undef FUNCTION_NAME__
8247:
8248:
8249: ADS1259_Calibration_Range_Test( &value, address, 1 );
8250:
8251: scratch = (uint32_t) value;
8252: for ( index = 0; index < byte_length; index++ )
8253: {
8254:     tx_buf[index + 2] = (uint8_t) ( scratch & 0xFF );
8255:     scratch = scratch >> 8;
8256: }
8257:
8258: error_code = AS_Registers_Write( board_id, address, byte_length, tx_buf, rx_buf );
8259: return error_code;
8260: }
8261:
8262: static int AS_Calibration_Value_Read( int board_id, size_t address, int32_t * value )
8263: {
8264:     int error_code;
8265:     uint32_t value_temp;
8266:     size_t index;
8267:     const size_t byte_length = 3;
8268:     uint8_t tx_buf[12] = { 0x00 };
8269:     uint8_t rx_buf[12] = { 0x00 };
8270:
8271:
8272:
8273: #define FUNCTION_NAME__ "AS_Calibration_Value_Read"
8274: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
8275:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
8276: #endif
8277: #undef FUNCTION_NAME__
8278:
8279: error_code = AS_Registers_Read( board_id, address, byte_length, tx_buf, rx_buf );
8280: if ( error_code < 0 ) goto AS_Calibration_Value_Read_Error_Exit;
8281: value_temp = 0;
8282:
8283: for ( index = 0; index < byte_length; index++ ) value_temp = ( value_temp << 8 ) + ( (uint32_t) rx_buf[byte_length - index + 2 - 1] );
8284:
8285:
8286: if ( 0 != ( 0xFF800000L & value_temp ) )
8287: {
8288:     value_temp = 0xFF800000L | value_temp;
8289: }
8290: *value = (int32_t) value_temp;
8291: return SUCCESS;
8292: AS_Calibration_Value_Read_Error_Exit:
```

```
8293:     return error_code;
8294: }
8295:
8296: static int CMD__AS_WReg( int board_id, int argc, char * argv[] )
8297: {
8298:     size_t count;
8299:     size_t index;
8300:     size_t address;
8301:     int error_code;
8302:     BOOL not_done;
8303:     uint8_t tx_buf[12] = { 0x00 };
8304:     uint8_t rx_buf[12] = { 0x00 };
8305:
8306:
8307:
8308: #define FUNCTION_NAME__ "CMD__AS_WReg"
8309: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
8310:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
8311: #endif
8312: #undef FUNCTION_NAME__
8313:
8314: if ( argc < 2 ) return -EC_SYNTAX;
8315:
8316:     index = 0;
8317:     if ( ( argv[index][0] >= '0' ) && ( argv[index][0] <= '9' ) )
8318:     {
8319:         address = (size_t) strtol( argv[index], NULL, 0 );
8320:     }
8321:     else
8322:     {
8323:         error_code = AS_Register_Name_To_Offset( argv[index] );
8324:         if ( error_code < 0 ) return -EC_NOT_FOUND;
8325:         address = (size_t) error_code;
8326:     }
8327:     index++;
8328:
8329:     count = 0;
8330:     not_done = true;
8331:     do
8332:     {
8333:         tx_buf[count + 2] = (uint8_t) strtol( argv[index], NULL, 0 );
8334:         count++;
8335:         index++;
8336:
8337:         if ( count >= AS_ADS1259_REGISTER_QTY )
8338:         {
8339:             not_done = false;
8340:             error_code = -EC_EXTRA_IGNORED;
8341:         }
8342:         if ( index >= argc ) not_done = false;
8343:     } while ( not_done );
8344:
8345:     error_code = AS_Registers_Write( board_id, address, count, tx_buf, rx_buf );
8346:     return error_code;
8347: }
8348:
8349: static int CMD__AS_Reg_Write_By_Name( int board_id, int argc, char * argv[] )
8350: {
8351:     size_t count;
8352:     size_t index;
8353:     size_t address;
8354:     size_t value;
8355:     int error_code;
8356:     uint8_t tx_buf[12] = { 0x00 };
8357:     uint8_t rx_buf[12] = { 0x00 };
8358:
8359:
8360:
```

```

8361: #define FUNCTION_NAME__ "CMD__AS_Reg_Write_By_Name"
8362: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
8363: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
8364: #endif
8365: #undef FUNCTION_NAME__
8366:
8367: if ( argc < 2 ) return -EC_SYNTAX;
8368:
8369: index = 0;
8370: while ( ( argc - index ) > 1 )
8371: {
8372: if ( (argc - index) > 1 )
8373: {
8374: if ( ( argv[index][0] >= '0' ) && ( argv[index][0] <= '9' ) )
8375: {
8376: address = (size_t) strtol( argv[index], NULL, 0 );
8377: }
8378: else
8379: {
8380: error_code = AS_Register_Name_To_Offset( argv[index] );
8381: if ( error_code < 0 ) return -EC_NOT_FOUND;
8382: address = (size_t) error_code;
8383: }
8384: index++;
8385:
8386: if ( ( argc - index ) > 0 )
8387: {
8388: value = (size_t) strtol( argv[index], NULL, 0 );
8389: tx_buf[2] = (uint8_t) value;
8390: count = 1;
8391: error_code = AS_Registers_Write( board_id, address, count, tx_buf, rx_buf );
8392: }
8393: else
8394: {
8395: return -EC_SYNTAX;
8396: }
8397: index++;
8398: }
8399: else
8400: {
8401: error_code = -EC_SYNTAX;
8402: }
8403: }
8404:
8405: if ( (argc - index) > 0 ) error_code = -EC_EXTRA_IGNORED;
8406: return error_code;
8407: }
8408:
8411: static int CMD__AS_Ofscal( int board_id, int argc, char * argv[] )
8412: {
8413: uint8_t tx_buf[1] = { 0x18 };
8414: (void) argc;
8415: (void) argv;
8416:
8417: #define FUNCTION_NAME__ "CMD__AS_Ofscal"
8418: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
8419: IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
8420: #endif
8421: #undef FUNCTION_NAME__
8422:
8423: return AS_Opcode_Write( board_id, tx_buf, 1 );
8424: }
8425:
8428: static int CMD__AS_Gancal( int board_id, int argc, char * argv[] )

```

```
8429: {
8430:     uint8_t tx_buf[1] = { 0x19 };
8431:     (void) argc;
8432:     (void) argv;
8433:
8434: #define FUNCTION_NAME__ "CMD__AS_Gancal"
8435: #if ( IO_TRACE_USE_AS_LOGGING == 1 )
8436:     IO_Trace_Log_Function( FUNCTION_NAME__, __LINE__, IO_TRACE_AS, board_id, 0 );
8437: #endif
8438: #undef FUNCTION_NAME__
8439:
8440:     return AS_Opcode_Write( board_id, tx_buf, 1 );
8441: }
8442:
8443: static int CMD__AS_Calibration_Gain( int board_id, int argc, char * argv[] )
8444: {
8445:     int32_t fsc;
8446:     int error_code;
8447:
8448:     if ( argc > 0 )
8449:     {
8450:         fsc = (int32_t) strtol( argv[0], NULL, 0 );
8451:         error_code = AS_Calibration_Value_Write( board_id, AS_AMS1259_FSC0, fsc );
8452:         if ( error_code < 0 ) return error_code;
8453:     }
8454:     else
8455:     {
8456:         error_code = AS_Calibration_Value_Read( board_id, AS_AMS1259_FSC0, &fsc );
8457:         if ( error_code < 0 ) return error_code;
8458:     }
8459: #if defined( __MSDOS__ )
8460:     printf( "Full Scale Calibration (FSC) = 0x%08lX, %ld\n", fsc, fsc );
8461: #else
8462:     printf( "Full Scale Calibration (FSC) = 0x%08X, %d\n", fsc, fsc );
8463: #endif
8464: }
8465: }
8466:     return SUCCESS;
8467: }
8468:
8469: static int CMD__AS_Calibration_Offset( int board_id, int argc, char * argv[] )
8470: {
8471:     int error_code;
8472:     int32_t ofc;
8473:
8474:     if ( argc > 0 )
8475:     {
8476:         ofc = (int32_t) strtol( argv[0], NULL, 0 );
8477:         error_code = AS_Calibration_Value_Write( board_id, AS_AMS1259_OFSC0, ofc );
8478:     }
8479:     else
8480:     {
8481:         error_code = AS_Calibration_Value_Read( board_id, AS_AMS1259_OFSC0, &ofc );
8482:         if ( error_code < 0 ) return error_code;
8483:     }
8484: #if defined( __MSDOS__ )
8485:     printf( "Offset Calibration (OFC) = 0x%08lX, %ld\n", ofc, ofc );
8486: #else
8487:     printf( "Offset Calibration (OFC) = 0x%08X, %d\n", ofc, ofc );
8488: #endif
8489: }
8490: }
8491:     return SUCCESS;
8492: }
8493:
8494: static int CMD__AS_Register_Save( int board_id, int argc, char * argv[] )
8495: {
8496:     int error_code;
```

```

8499: FILE * fd;
8500: uint8_t tx_buf[12] = { 0x00 };
8501: uint8_t rx_buf[12] = { 0x00 };
8502:
8503: if ( argc < 1 ) return -EC_SYNTAX;
8504:
8505: error_code = AS_Registers_Read( board_id, AS_AM2315_CONFIG0, AS_AM2315_REGISTER_QTY,
tx_buf, rx_buf );
8506: if ( error_code < 0 ) return error_code;
8507:
8508: fd = fopen( argv[0], "w" );
8509: if ( NULL != fd )
8510: {
8511: void * data_pointer = (void *) &(rx_buf[2]);
8512: fwrite( data_pointer, AS_AM2315_REGISTER_QTY, 1, fd );
8513: fclose( fd );
8514: }
8515: return SUCCESS;
8516: }
8517: #if(0)
8518:
8519: static void AS_Register_Defaults( uint8_t * buf, size_t count )
8520: {
8521: if ( count < AS_AM2315_REGISTER_QTY ) return;
8522: buf[AS_AM2315_CONFIG0] = 0x85;
8523: buf[AS_AM2315_CONFIG1] = 0x10;
8524: buf[AS_AM2315_CONFIG2] = 0x00;
8525: buf[AS_AM2315_OF0] = 0x00;
8526: buf[AS_AM2315_OF1] = 0x00;
8527: buf[AS_AM2315_OF2] = 0x00;
8528: buf[AS_AM2315_FSC0] = 0x00;
8529: buf[AS_AM2315_FSC1] = 0x00;
8530: buf[AS_AM2315_FSC2] = 0x40;
8531:
8532: }
8533: }
8534: #endif
8535:
8536: static int CMD_AM2315_Register_Load( int board_id, int argc, char * argv[] )
8537: {
8538: int error_code;
8539: FILE * fd;
8540: void * data_pointer = NULL;
8541: uint8_t tx_buf[12] = { 0x00 };
8542: uint8_t rx_buf[12] = { 0x00 };
8543: struct board_dataset * dataset;
8544:
8545: dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
8546: error_code = SUCCESS;
8547:
8548: if ( 0 == argc )
8549: {
8550: size_t index;
8551: for ( index = 0; index < AS_AM2315_REGISTER_QTY; index ++ )
8552: {
8553: tx_buf[index + 2] = dataset->am2315.reg[index];
8554: }
8555:
8556: }
8557:
8558: error_code = AS_Registers_Write( board_id, AS_AM2315_CONFIG0,
AS_AM2315_REGISTER_QTY, tx_buf, rx_buf );
8559: }
8560: else
8561: {
8562: fd = fopen( argv[0], "r" );
8563: if ( NULL == fd )
8564: {

```

```
8565:     error_code = -EC_FILE_ERROR;
8566: }
8567: else
8568: {
8569:     size_t file_size;
8570:
8571:     fseek( fd, 0L, SEEK_END );
8572:     file_size = (size_t) ftell( fd );
8573:     fseek( fd, 0L, SEEK_SET );
8574:     if ( file_size != AS_AM2315_REGISTER_QTY )
8575:     {
8576:         error_code = -EC_FILE_ERROR_SIZE;
8577:     }
8578: else
8579: {
8580:     data_pointer = (void *) &(tx_buf[2]);
8581:     fread( data_pointer, AS_AM2315_REGISTER_QTY, 1, fd );
8582:     error_code = AS_Registers_Write( board_id, AS_AM2315_CONFIG0,
AS_AM2315_REGISTER_QTY, tx_buf, rx_buf );
8583: }
8584: fclose( fd );
8585: }
8586: }
8587: return error_code;
8588: }
8589:
8590:
8593: static int CMD__AS_Parameter( int board_id, int argc, char * argv[] )
8594: {
8595:     int      error_code = SUCCESS;
8596:     int      argc_new;
8597:     char **  argv_new;
8598:     int      index;
8599:     uint8_t   tx_buf[12] = { 0x00 };
8600:     uint8_t   rx_buf[12] = { 0x00 };
8601:
8602:     if ( ( argc < 1 ) || ( NULL == argv ) )
8603:     {
8604:         argv_new = &(argv[1]);
8605:         argc_new = argc - 1;
8606:         printf( "Calibration offset: " );
8607:         error_code = CMD__AS_Calibration_Offset( board_id, argc_new, argv_new );
8608:         if ( error_code < 0 ) goto CMD__AS_PARAMETER_END;
8609:         printf( "Calibration gain: " );
8610:         error_code = CMD__AS_Calibration_Gain( board_id, argc_new, argv_new );
8611:         if ( error_code < 0 ) goto CMD__AS_PARAMETER_END;
8612:         printf( "Registers:\n" );
8613:         for ( index = 0; index < AS_AM2315_REGISTER_QTY; index++ )
8614:         {
8615:             error_code = CMD__AS_RReg( board_id, 1, &(as_register_name[index]) );
8616:             if ( error_code < 0 ) goto CMD__AS_PARAMETER_END;
8617:         }
8618:
8619:         printf( "Bit String Parameters:\n" );
8620:         index = 0;
8621:         while ( NULL != as_bit_string[index].name )
8622:         {
8623:             struct bit_string_info * bsd;
8624:             size_t i;
8625:             uint8_t mask = 0;
8626:             bsd = &(as_bit_string[index]);
8627:             error_code = AS_Registers_Read( board_id, bsd->register_offset, 1, tx_buf, rx_buf );
8628:             if ( error_code < 0 ) goto CMD__AS_PARAMETER_END;
8629:         }
```

```

8630:     for ( i = 0; i < bsd->bit_width; i++ ) mask = ( mask << 1 ) | 0x01;
8631:     rx_buf[2] = rx_buf[2] >> bsd->bit_offset;
8632:     rx_buf[2] = rx_buf[2] & mask;
8633:     printf( " %8s = %d\n", bsd->name, (int) rx_buf[2] );
8634:
8635:     index++;
8636:   }
8637: }
8638: if ( argc >= 2 )
8639: {
8640:   if ( 0 == strcmpi( "ofc", argv[0] ) )
8641:   {
8642:     argv_new = &(argv[1]);
8643:     argc_new = argc - 1;
8644:     error_code = CMD__AS_Calibration_Offset( board_id, argc_new, argv_new );
8645:   }
8646: else if ( 0 == strcmpi( "fsc", argv[0] ) )
8647: {
8648:   argv_new = &(argv[1]);
8649:   argc_new = argc - 1;
8650:   error_code = CMD__AS_Calibration_Gain( board_id, argc_new, argv_new );
8651: }
8652: else if ( ( index = AS_Register_Name_To_Offset( argv[0] ) ) >= 0 )
8653: {
8654:   error_code = CMD__AS_Reg_Write_By_Name( board_id, argc, argv );
8655: }
8656: else if ( ( index = Bit_String_Index_By_Name( as_bit_string, argv[0] ) ) >= 0 )
8657: {
8658:   struct bit_string_info * bsd;
8659:   size_t temp;
8660:   uint8_t mask = 0;
8661:   bsd = &(as_bit_string[index]);
8662:   error_code = AS_Registers_Read( board_id, bsd->register_offset, 1, tx_buf, rx_buf );
8663:
8664:   for ( temp = 0; temp < bsd->bit_width; temp++ ) mask = ( mask << 1 ) | 0x01;
8665:   temp = (size_t) strtol( argv[1], NULL, 0 );
8666:   temp = temp & mask;
8667:   temp = temp << bsd->bit_offset;
8668:   rx_buf[2] = rx_buf[2] & ~( mask << bsd->bit_offset );
8669:   tx_buf[2] = rx_buf[2] | ( (uint8_t) temp );
8670:   error_code = AS_Registers_Write( board_id, bsd->register_offset, 1, tx_buf, rx_buf );
8671: }
8672: else
8673: {
8674:   error_code = -EC_NOT_FOUND;
8675: }
8676: }
8677: else
8678: {
8679:   if ( 0 == strcmpi( "ofc", argv[0] ) )
8680:   {
8681:     argv_new = &(argv[1]);
8682:     argc_new = argc - 1;
8683:     error_code = CMD__AS_Calibration_Offset( board_id, argc_new, argv_new );
8684:   }
8685: else if ( 0 == strcmpi( "fsc", argv[0] ) )
8686: {
8687:   argv_new = &(argv[1]);
8688:   argc_new = argc - 1;
8689:   error_code = CMD__AS_Calibration_Gain( board_id, argc_new, argv_new );
8690: }
8691: else if ( ( index = AS_Register_Name_To_Offset( argv[0] ) ) >= 0 )
8692: {
8693:   error_code = CMD__AS_RReg( board_id, argc, argv );

```

```

8694: }
8695: else if ( ( index = Bit_String_Index_By_Name( as_bit_string, argv[0] ) ) >= 0 )
8696: {
8697:     struct bit_string_info * bsd;
8698:     size_t i;
8699:     uint8_t mask = 0;
8700:     bsd = &(as_bit_string[index]);
8701:     error_code = AS_Registers_Read( board_id, bsd->register_offset, 1, tx_buf, rx_buf );
8702:     if ( error_code < 0 ) goto CMD__AS_PARAMETER_END;
8703:
8704:     for ( i = 0; i < bsd->bit_width; i++ ) mask = ( mask << 1 ) | 0x01;
8705:     rx_buf[2] = rx_buf[2] >> bsd->bit_offset;
8706:     rx_buf[2] = rx_buf[2] & mask;
8707:     printf( "%8s = %d\n", bsd->name, (int) rx_buf[2] );
8708: }
8709: else
8710: {
8711:     error_code = -EC_NOT_FOUND;
8712: }
8713: }
8714: CMD__AS_PARAMETER_END:
8715: return error_code;
8716: }
8717:
8718:
8722: static struct command_line_board cmd_as[] =
8723: {
8724:     { NULL, CMD__AS_Wakeup,      "wakeup",      "ADS1259 exit sleep mode" },
8725:     { NULL, CMD__AS_Sleep,       "sleep",       "ADS1259 enter sleep mode" },
8726:     { NULL, CMD__AS_Reset,      "reset",       "ADS1259 register and filter reset" },
8727:     { NULL, CMD__AS_Start,      "start",       "ADS1259 start conversion(s)" },
8728:     { NULL, CMD__AS_Stop,       "stop",        "ADS1259 stop conversions" },
8729:     { NULL, CMD__AS_Rdatac,     "rdatac",      "ADS1259 set read data continuous mode" },
8730:     { NULL, CMD__AS_Sdatac,     "sdatac",      "ADS1259 stop read data continuous mode" },
8731:     { NULL, CMD__AS_Rdata,      "rdata",       "ADS1259 read data in stop continuous mode" },
8732:     { NULL, CMD__AS_RReg,       "rreg",        "ADS1259 read N registers, rreg <address>" },
8733:     { NULL, CMD__AS_WReg,       "wreg",        "ADS1259 write N registers, wreg <address> <value>" },
8734:     { NULL, CMD__AS_Ofscal,     "ofscal",      "ADS1259 perform offset calibration, apply zero input" },
8735:     { NULL, CMD__AS_Gancal,     "gancal",      "ADS1259 perform full-scale calibration, apply full-scale" },
8736:     { NULL, CMD__AS_Reg_Write_By_Name, "wrn",      "Write one or more registers by name, wrn <name> <value>" },
8737:     { NULL, CMD__AS_Calibration_Gain, "fsc",       "read/write fullscale calibration" },
8738:     { NULL, CMD__AS_Calibration_Offset, "ofc",       "read/write offset calibration" },
8739:     { NULL, CMD__AS_Register_Save,   "rsave",     "save all registers to a file" },
8740:     { NULL, CMD__AS_Register_Load,   "rload",     "restore all registers from file" },
8741:     { NULL, CMD__AS_Parameter,     "param",     "read/write parameter" },
8742:     { NULL, NULL, NULL, NULL } },
8743: };
8744:
8750: int Command_Line_Analog_Stick( int board_id, int argc, char* argv[] )
8751: {
8752:     int error_code;
8753:     int index;
8754:     int argc_new;
8755:     char ** argv_new;
8756:
8757:     error_code = -EC_SYNTAX;
8758:
8759:     if ( argc < 1 ) return -EC_NOT_FOUND;
8760:

```

```

8761: index = 0;
8762: while ( NULL != cmd_as[index].cmd_fnc )
8763: {
8764:   if ( 0 == strcasecmp( cmd_as[index].name, argv[0] ) )
8765:   {
8766:     argv_new = &(argv[1]);
8767:     argc_new = argc - 1;
8768:     if ( 0 == argc_new ) argv_new = NULL;
8769:
8770:   }
8771:   struct board_dataset * dataset;
8772:   dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
8773:   if ( IO_TRACE_START_AS == dataset->trace.start )
8774:   {
8775:     dataset->trace.active = true;
8776:   }
8777: }
8778:
8779: error_code = (* cmd_as[index].cmd_fnc )( board_id, argc_new, argv_new );
8780: break;
8781: }
8782: index++;
8783: }
8784: return error_code;
8785: }
8786:
8787:
8788:
8789:
8790:
8791:
8792:
8793:
8794: static int CMD__FRAM_Dump( int board_id, int argc, char * argv[] )
8795: {
8796:   uint16_t address;
8797:   uint16_t length;
8798:
8799:   if ( argc < 1 ) return -EC_PARAMETER;
8800:
8801:   address = (uint16_t) strtol( argv[0], NULL, 0 );
8802:   if ( address > ( FRAM_DENSITY_BYTES - 1 ) ) return -EC_FRAM_ADDRESS_RANGE;
8803:
8804:   if ( argc < 2 ) length = HEX_DUMP_BYTES_PER_LINE;
8805:   else           length = (uint16_t) strtol( argv[1], NULL, 0 );
8806:
8807:   if ( ( address + length ) > FRAM_DENSITY_BYTES ) length = FRAM_DENSITY_BYTES - address;
8808:
8809:   if ( FRAM_Report( board_id, address, length, stdout ) );
8810:
8811:   return FRAM_Report( board_id, address, length, stdout );
8812:
8813: }
8814:
8815: }
8816:
8817: static int CMD__FRAM_Save( int board_id, int argc, char * argv[] )
8818: {
8819:   int error_code;
8820:   uint16_t address;
8821:   size_t length;
8822:   FILE * out;
8823:
8824:   if ( argc < 3 ) return -EC_PARAMETER;
8825:
8826:   address = (uint16_t) strtol( argv[0], NULL, 0 );
8827:   if ( address > ( FRAM_DENSITY_BYTES - 1 ) ) return -EC_FRAM_ADDRESS_RANGE;
8828:
8829:   length = (uint16_t) strtol( argv[1], NULL, 0 );
8830:
8831:   if ( FRAM_Report( board_id, address, length, out ) );
8832:
8833:   return FRAM_Report( board_id, address, length, out );
8834:
```

```
8835: if ( (address + length - 1) > ( FRAM_DENSITY_BYTES - 1 ) ) return  
-EC_FRAM_ADDRESS_RANGE;  
8836: out = fopen( argv[2], "w" );  
8837: error_code = FRAM_Memory_To_File( board_id, address, length, out );  
8838: fclose( out );  
8839: return error_code;  
8840: }  
8841:  
8842: static int CMD__FRAM_Load( int board_id, int argc, char * argv[] )  
8843: {  
8844: int error_code;  
8845: uint16_t address;  
8846:  
8847: FILE * out;  
8848:  
8849: if ( argc < 2 ) return -EC_PARAMETER;  
8850:  
8851: address = (uint16_t) strtol( argv[0], NULL, 0 );  
8852: if ( address > ( FRAM_DENSITY_BYTES - 1 ) ) return -EC_FRAM_ADDRESS_RANGE;  
8853: out = fopen( argv[2], "r" );  
8854: error_code = FRAM_File_To_Memory( board_id, address, 0 , out );  
8855: fclose( out );  
8856: return error_code;  
8857: }  
8858:  
8859: static int CMD__FRAM_Write( int board_id, int argc, char * argv[] )  
8860: {  
8861: int error_code;  
8862: size_t count;  
8863: uint16_t address;  
8864: struct board_dataset * dataset;  
8865: dataset = ( struct board_dataset * ) board_definition[board_id].dataset;  
8866:  
8867: if ( argc < 2 ) return -EC_PARAMETER;  
8868:  
8869: address = (uint16_t) strtol( argv[0], NULL, 0 );  
8870: if ( address > ( FRAM_DENSITY_BYTES - 1 ) ) return -EC_FRAM_ADDRESS_RANGE;  
8871:  
8872: error_code = CMD__SPI_Data_Interpreter( 1,  
8873: argc,  
8874: argv,  
8875: FRAM_BLOCK_SIZE,  
8876: &count,  
8877: dataset->spi.fram_block  
8878: );  
8879: if ( SUCCESS != error_code ) goto CMD__FRAM_Write_Error_Exit;  
8880: error_code = FRAM__Memory_Write( board_id, address, count, dataset->spi.fram_block );  
8881: CMD__FRAM_Write_Error_Exit:  
8882: return error_code;  
8883: }  
8884:  
8885: static int CMD__FRAM_Init( int board_id, int argc, char * argv[] )  
8886: {  
8887: int error_code;  
8888:  
8889: size_t tx_size_in_bytes;  
8890: struct board_dataset * dataset;  
8891: dataset = ( struct board_dataset * ) board_definition[board_id].dataset;  
8892:  
8893:  
8894: if ( argc < 1 )  
8895: {  
8896: error_code = FRAM_Set( board_id, 0, NULL );  
8897:  
8898: }  
8899:
```

```
8913: else
8914: {
8915:     CMD__SPI_Data_Interpreter( 0,
8916:         argc,
8917:         argv,
8918:         FRAM_BLOCK_SIZE,
8919:         &tx_size_in_bytes,
8920:         dataset->spi.fram_block
8921:     );
8922:
8923: #if defined( IDI_FRAM_PRINT_DEBUG )
8924:
8925: {
8926:     size_t display_count;
8927:
8928:     if ( tx_size_in_bytes > HEX_DUMP_BYTES_PER_LINE ) display_count =
HEX_DUMP_BYTES_PER_LINE;
8929:     else
8930:             display_count = tx_size_in_bytes;
8931:     Hex_Dump_Line( 0, display_count, dataset->spi.fram_block, stdout );
8932: }
8933: #endif
8934:     error_code = FRAM_Set( board_id, tx_size_in_bytes, dataset->spi.fram_block );
8935: }
8936: return error_code;
8937: }
8938:
8944: static int CMD__FRAM_WREN( int board_id, int argc, char * argv[] )
8945: {
8946:     (void) argc;
8947:     (void) argv;
8948:     return FRAM__Write_Enable_Latch_Set( board_id );
8949: }
8950:
8956: static int CMD__FRAM_WRDI( int board_id, int argc, char * argv[] )
8957: {
8958:     (void) argc;
8959:     (void) argv;
8960:     return FRAM__Write_Disable( board_id );
8961: }
8962:
8968: static int CMD__FRAM_RDSR( int board_id, int argc, char * argv[] )
8969: {
8970:     int error_code;
8971:     uint8_t status;
8972:     (void) argc;
8973:     (void) argv;
8974:     error_code = FRAM__Read_Status_Register( board_id, &status );
8975:     printf( "FRAM STATUS: 0x%02X\n", ((int) status) );
8976:     return error_code;
8977: }
8978:
8984: static int CMD__FRAM_WRSR( int board_id, int argc, char * argv[] )
8985: {
8986:
8987:     uint8_t status;
8988:
8989:     if ( argc < 1 ) return -EC_PARAMETER;
8990:     status = (uint8_t) strtol( argv[0], NULL, 0 );
8991:     return FRAM__Write_Status_Register( board_id, status );
8992: }
8993:
8999: static int CMD__FRAM_RDID( int board_id, int argc, char * argv[] )
9000: {
```

```
9001: uint32_t id;
9002: (void) argc;
9003: (void) argv;
9004: FRAM_Read_ID( board_id, &id );
9005: #if defined( __MSDOS__ )
9006: printf( "FRAM ID: 0x%08lX\n", id );
9007: #else
9008: printf( "FRAM ID: 0x%08X\n", id );
9009: #endif
9010: return SUCCESS;
9011: }
9012:
9016: static struct command_line_board cmd_fram[] =
9017: {
9018: { NULL, CMD_FRAM_Dump, "dump", "params: <address> <length>" },
9019: { NULL, CMD_FRAM_Save, "save", "params: <address> <length> <binary destination
file>" },
9020: { NULL, CMD_FRAM_Load, "load", "params: <address> <binary source file name>" },
9021: { NULL, CMD_FRAM_Write, "write", "params: <address> <data list:
bytes/characters/strings>" },
9022: { NULL, CMD_FRAM_Init, "init", "params: [byte/character] [byte/character] ..." },
9023: { NULL, CMD_FRAM_WREN, "wren", "WRite Enable Latch Set" },
9024: { NULL, CMD_FRAM_WRDI, "wrdi", "WRite DIable" },
9025: { NULL, CMD_FRAM_RDSR, "rdsr", "ReaD Status Register" },
9026: { NULL, CMD_FRAM_WRSR, "wrsr", "WRite Status Register. Params: <status>" },
9027: { NULL, CMD_FRAM_RDID, "rdid", "ReaD ID Register" },
9028: { NULL, NULL, NULL, NULL } },
9029: };
9030:
9036: int Command_Line_FRAM( int board_id, int argc, char* argv[] )
9037: {
9038: int error_code;
9039: int index;
9040: int argc_new;
9041: char ** argv_new;
9042:
9043: error_code = -EC_SYNTAX;
9044:
9045: if ( argc < 1 ) return -EC_NOT_FOUND;
9046:
9047: index = 0;
9048: while ( NULL != cmd_fram[index].cmd_fnc )
9049: {
9050: if ( 0 == strcasecmp( cmd_fram[index].name, argv[0] ) )
9051: {
9052: argv_new = &(argv[1]);
9053: argc_new = argc - 1;
9054: if ( 0 == argc_new ) argv_new = NULL;
9055:
9056: {
9057: struct board_dataset * dataset;
9058: dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
9059: if ( IO_TRACE_START_FRAM == dataset->trace.start )
9060: {
9061: dataset->trace.active = true;
9062: }
9063: }
9064:
9065: error_code = (* cmd_fram[index].cmd_fnc )( board_id, argc_new, argv_new );
9066: break;
9067: }
9068: index++;
9069: }
9070: return error_code;
```

```

9071: }
9072:
9073:
9074: typedef enum
9075: {
9076:     TEST_STATE_INIT      = 0x000,
9077:     TEST_STATE_READY     = 0x100,
9078:     TEST_STATE_ACTIVE    = 0x200,
9079:     TEST_STATE_COMPLETE   = 0xF00,
9080:     TEST_STATE_QUIT_ERROR = 0xF10,
9081:     TEST_STATE_QUIT_USER  = 0xF20,
9082:     TEST_STATE_QUIT      = 0xFFFF
9083: } TEST_STATE_ENUM;
9084:
9085:
9086:
9087:
9088:
9089:
9090:
9091:
9092:
9101: int IDO_DOUT_ID_Get( int board_id, uint16_t * id )
9102: {
9103:     uint8_t lsb, msb;
9104:     struct board_dataset * dataset;
9105:     dataset = (struct board_dataset *) board_definition[board_id].dataset;
9106:
9107:     if ( MODE_JUMPERS_0 == dataset->mode_jumpers )
9108:     {
9109:         return -EC_MODE_LEGACY;
9110:     }
9111:
9112:     IO_Read_U8( board_id, __LINE__, IDO_ID_LSB, &lsb );
9113:     IO_Read_U8( board_id, __LINE__, IDO_ID_MSB, &msb );
9114:     *id = ( ((uint16_t) msb) << 8 ) | ((uint16_t) lsb);
9115: #if defined( ID_ALWAYS_REPORT_AS_GOOD )
9116:     *id = ID_DOUT;
9117: #endif
9118:     return SUCCESS;
9119: }
9120:
9128: BOOL IDO_DOUT_IsNotPresent( int board_id )
9129: {
9130:     uint16_t id;
9131:     IDO_DOUT_ID_Get( board_id, &id );
9132:     if (ID_DOUT == id) return 0;
9133:     return 1;
9134: }
9135:
9145: int IDO_DO_Channel_Get( int board_id, size_t channel, BOOL * value )
9146: {
9147:     size_t group;
9148:     size_t bit;
9149:     uint8_t reg_value;
9150:
9151:     group = channel >> IDO_DO_SHIFT_RIGHT;
9152:     bit   = channel - group * IDO_DO_GROUP_SIZE;
9153:
9154:     IO_Read_U8( board_id, __LINE__, IDO_DO_GROUP0 + group , &reg_value );
9155:
9156:     if ( 0 != ( reg_value & ( 0x01 << bit ) ) ) *value = true;
9157:     else                                *value = false;
9158:
```

```
9159:     return SUCCESS;
9160: }
9161:
9171: int IDO_DO_Channel_Set( int board_id, size_t channel, BOOL value )
9172: {
9173:     size_t group;
9174:     size_t bit;
9175:     uint8_t reg_value;
9176:     uint8_t mask;
9177:
9178:     group = channel >> IDO_DO_SHIFT_RIGHT;
9179:     bit = channel - group * IDO_DO_GROUP_SIZE;
9180:
9181:     IO_Read_U8( board_id, __LINE__, IDO_DO_GROUP0 + group , &reg_value );
9182:     mask = 0x01 << bit;
9183:     if ( true == value ) reg_value |= mask;
9184:     else     reg_value &= ~mask;
9185:
9186:     IO_Write_U8( board_id, __LINE__, IDO_DO_GROUP0 + group , reg_value );
9187:     return SUCCESS;
9188: }
9189:
9199: int IDO_DO_Group_Get( int board_id, size_t group, uint8_t * value )
9200: {
9201:     IO_Read_U8( board_id, __LINE__, IDO_DO_GROUP0 + group , value );
9202:     return SUCCESS;
9203: }
9204:
9214: int IDO_DO_Group_Set( int board_id, size_t group, uint8_t value )
9215: {
9216:     IO_Write_U8( board_id, __LINE__, IDO_DO_GROUP0 + group , value );
9217:     return SUCCESS;
9218: }
9219:
9220:
9224: static int Print_BytE_List( const char * prefix_string,
9225:                               int argc,
9226:                               char ** argv,
9227:                               size_t list_count,
9228:                               uint8_t * list,
9229:                               FILE * out
9230:                               )
9231: {
9232:     enum { MODE_NONE = 0, MODE_BINARY = 1, MODE_HEX = 2, MODE_ALL = 3 } mode_out;
9233:     int group;
9234:
9235:     mode_out = MODE_ALL;
9236:     if ( NULL == argv )
9237:     {
9238:         mode_out = MODE_ALL;
9239:     }
9240:     else
9241:     {
9242:         int index;
9243:         for ( index = 0; index < argc; index++ )
9244:         {
9245:             if ( 0 == strcmpi( "binary", argv[index] ) ) mode_out |= MODE_BINARY;
9246:             else if ( 0 == strcmpi( "group", argv[index] ) ) mode_out |= MODE_HEX;
9247:             else if ( 0 == strcmpi( "hex", argv[index] ) ) mode_out |= MODE_HEX;
9248:             else mode_out |= MODE_ALL;
9249:         }
9250:     }
9251:
9252:     if ( MODE_BINARY == ( mode_out & MODE_BINARY ) )
```

```

9253: {
9254:     int cp;
9255:     int channel;
9256:     uint8_t mask;
9257:     char message[64];
9258:
9259:     cp = 0;
9260:     group = 0;
9261:     for ( group = 0; group < list_count; group++ )
9262:     {
9263:         mask = 0x01;
9264:         for ( channel = 0; channel < 8; channel++ )
9265:         {
9266:             message[cp++] = !(list[group] & mask) ? '1' : '0';
9267:             mask = mask << 1;
9268:         }
9269:         message[cp++] = ' ';
9270:     }
9271:     message[cp] = '\0';
9272:     fprintf( out, "%s: %s\n", prefix_string, message );
9273: }
9274:
9275: if ( MODE_HEX == ( mode_out & MODE_HEX ) )
9276: {
9277:     fprintf( out, "%s:", prefix_string );
9278:     for ( group = 0; group < list_count; group++ ) fprintf( out, " %02X", list[group] );
9279:     fprintf( out, "\n" );
9280: }
9281:
9282: return SUCCESS;
9283: }
9284:
9285: #if ( IDO_DO_GROUP_QTY == IDI_DIN_GROUP_QTY )
9286:
9287:
9288:
9289: static int IDO48_IDI48_Loopback_Test( int di_id, int do_id, uint8_t * di_list, uint8_t
* do_list, size_t count )
9290: {
9291:     int group;
9292:     for( group = 0; group < count; group++ )
9293:     {
9294:         IDO_DO_Group_Get( do_id, group, &(do_list[group]) );
9295:         IDI_DIN_Group_Get( di_id, group, &(di_list[group]) );
9296:         if ( di_list[group] != do_list[group] ) return -group;
9297:     }
9298:     return SUCCESS;
9299: }
9300:
9301:
9302: }
9303:
9304: static int CMD__IDO48_IDI48_Loopback( int board_id, int argc, char * argv[] )
9305: {
9306:     int error_code;
9307:     struct ido_dataset * dataset;
9308:     FILE * out;
9309:     char * file_name;
9310:     TEST_STATE_ENUM state;
9311:     size_t group;
9312:     size_t channel;
9313:     BOOL di_value;
9314:     BOOL do_value;
9315: #if defined( __MSDOS__ )
9316:     size_t wait_ms;
9317: #endif
9318:     uint8_t do_list[IDO_DO_GROUP_QTY];

```

```
9321: uint8_t    di_list[IDI_DIN_GROUP_QTY];
9322: uint8_t    dog[IDO_DO_GROUP_QTY];
9323: char *     mode_string[] = { "binary" };
9324: char       out_default[] = { "stdout" };
9325: char *     msg;
9326:
9327: state = TEST_STATE_INIT;
9328: dataset = NULL;
9329: switch( board_id )
9330: {
9331:     case ID_IDO48:
9332:         dataset = ( struct ido_dataset * ) board_definition[board_id].dataset;
9333:         break;
9334:     default:
9335:         return -EC_BOARD_TYPE;
9336:     }
9337:
9338: #if defined( __MSDOS__ )
9339: wait_ms = 100;
9340: #endif
9341: out      = stdout;
9342: file_name = out_default;
9343: if ( argc > 0 )
9344: {
9345:     size_t wi;
9346:     char open_method[4];
9347:     wi = 0;
9348:     strcpy( open_method, "wt" );
9349:     if ( ( 0 == strcasecmp( "--a", argv[wi] ) ) || 
9350:          ( 0 == strcasecmp( "--append", argv[wi] ) ) )
9351:     {
9352:         wi++;
9353:         strcpy( open_method, "at" );
9354:     }
9355:
9356:     if ( argc > wi )
9357:     {
9358:         out = fopen( argv[wi], open_method );
9359:         file_name = argv[wi];
9360:     }
9361: }
9362:
9363:
9364:
9365: msg = &(dataset->message[0]);
9366: sprintf( msg, "===== " );
9367: Print_Multiple( msg, out );
9368: Time_Current_String( msg, MESSAGE_SIZE );
9369: Print_Multiple( msg, out );
9370:
9371: sprintf( msg, "\nIDO48-IDI48 loopback test: %s", file_name );
9372: Print_Multiple( msg, out );
9373:
9374: for( group = 0; group < IDO_DO_GROUP_QTY; group++ )
9375: {
9376:     do_list[group] = 0x00;
9377:     IDO_DO_Group_Set( board_id, group, do_list[group] );
9378: #if defined( __MSDOS__ )
9379:     delay( wait_ms );
9380: #endif
9381:     IDI_DIN_Group_Get( ID_IDI48, group, &(di_list[group]) );
9382: }
9383: error_code = IDO48_IDI48_Loopback_Test( ID_IDI48, board_id, di_list, do_list,
```

```
IDO_DO_GROUP_QTY );
9388: if ( error_code < 0 )
9389: {
9390:   state = TEST_STATE_QUIT;
9391: }
9392:
9393: state = TEST_STATE_ACTIVE;
9394: channel = 0;
9395: while ( TEST_STATE_ACTIVE == state )
9396: {
9397:
9398:
9399:   do_value = true;
9400:   IDO_DO_Channel_Set( board_id, channel, do_value );
9401: #if defined( __MSDOS__ )
9402:   delay( wait_ms );
9403: #endif
9404:   IDI_DIN_Channel_Get( ID_IDI48, channel, &di_value );
9405:
9406:   if ( do_value != di_value )
9407:   {
9408:     state = TEST_STATE_QUIT_ERROR;
9409:   }
9410:   error_code = IDO48_IDI48_Loopback_Test( ID_IDI48, board_id, di_list, do_list,
IDO_DO_GROUP_QTY );
9411:   if ( error_code < 0 )
9412:   {
9413:     state = TEST_STATE_QUIT_ERROR;
9414:   }
9415:
9416:
9417:   do_value = false;
9418:   IDO_DO_Channel_Set( board_id, channel, do_value );
9419: #if defined( __MSDOS__ )
9420:   delay( wait_ms );
9421: #endif
9422:   IDI_DIN_Channel_Get( ID_IDI48, channel, &di_value );
9423:
9424:   if ( do_value != di_value )
9425:   {
9426:     state = TEST_STATE_QUIT_ERROR;
9427:   }
9428:   error_code = IDO48_IDI48_Loopback_Test( ID_IDI48, board_id, di_list, do_list,
IDO_DO_GROUP_QTY );
9429:   if ( error_code < 0 )
9430:   {
9431:     state = TEST_STATE_QUIT_ERROR;
9432:   }
9433:
9434:
9435:   {
9436:     size_t index;
9437:     uint8_t do_group[IDO_DO_GROUP_QTY];
9438:
9439:     for ( index = 0; index < IDO_DO_GROUP_QTY; index++ )
9440:     {
9441:       do_group[index] = 0xFF;
9442:       IDO_DO_Group_Set( board_id, index, do_group[index] );
9443:     }
9444: #if defined( __MSDOS__ )
9445:   delay( wait_ms );
9446: #endif
9447:   error_code = IDO48_IDI48_Loopback_Test( ID_IDI48, board_id, di_list, do_list,
IDO_DO_GROUP_QTY );
```

```
9448:     if ( error_code < 0 )
9449:     {
9450:         sprintf( msg, "All high fail" );
9451:         Print_Multiple( msg, out );
9452:         state = TEST_STATE_QUIT_ERROR;
9453:     }
9454: else
9455: {
9456:     for ( index = 0; index < IDO_DO_GROUP_QTY; index++ )
9457:     {
9458:         do_group[index] = 0x00;
9459:         IDO_DO_Group_Set( board_id, index, do_group[index] );
9460:     }
9461: }
9462: }
9463:
9464:
9465: if ( true == Character_Get( NULL ) ) state = TEST_STATE_QUIT_USER;
9466:
9467: if ( TEST_STATE_ACTIVE == state )
9468: {
9469:     channel++;
9470:     sprintf( msg, "." );
9471:     Print_Multiple( msg, out );
9472:     if ( channel >= IDO_DO_QTY ) state = TEST_STATE_COMPLETE;
9473: }
9474: }
9475:
9476:
9477: sprintf( msg, "[PATTERN_INVERT]" );
9478: Print_Multiple( msg, out );
9479: channel = 0;
9480: if ( TEST_STATE_COMPLETE == state ) state = TEST_STATE_ACTIVE;
9481: while ( TEST_STATE_ACTIVE == state )
9482: {
9483:     {
9484:         size_t index;
9485:
9486:         for ( index = 0; index < IDO_DO_GROUP_QTY; index++ )
9487:         {
9488:             IDO_DO_Group_Set( board_id, index, 0xFF );
9489:         }
9490: #if defined( __MSDOS__ )
9491:     delay( wait_ms );
9492: #endif
9493:     error_code = IDO48_IDI48_Loopback_Test( ID_IDI48, board_id, di_list, do_list,
IDO_DO_GROUP_QTY );
9494:     if ( error_code < 0 )
9495:     {
9496:         sprintf( msg, "All high fail" );
9497:         Print_Multiple( msg, out );
9498:         state = TEST_STATE_QUIT_ERROR;
9499:     }
9500: }
9501:
9502:
9503: do_value = false;
9504: IDO_DO_Channel_Set( board_id, channel, do_value );
9505: #if defined( __MSDOS__ )
9506: delay( wait_ms );
9507: #endif
9508: IDI_DIN_Channel_Get( ID_IDI48, channel, &di_value );
9509:
9510: if ( do_value != di_value )
```

```
9511:  {
9512:    state = TEST_STATE_QUIT_ERROR;
9513:  }
9514:  error_code = IDO48_IDI48_Loopback_Test( ID_IDI48, board_id, di_list, do_list,
IDO_DO_GROUP_QTY );
9515:  if ( error_code < 0 )
9516:  {
9517:    state = TEST_STATE_QUIT_ERROR;
9518:  }
9519:
9520:
9521:  do_value = true;
9522:  IDO_DO_Channel_Set( board_id, channel, do_value );
9523: #if defined( __MSDOS__ )
9524:  delay( wait_ms );
9525: #endif
9526:  IDI_DIN_Channel_Get( ID_IDI48, channel, &di_value );
9527:
9528:  if ( do_value != di_value )
9529:  {
9530:    state = TEST_STATE_QUIT_ERROR;
9531:  }
9532:  error_code = IDO48_IDI48_Loopback_Test( ID_IDI48, board_id, di_list, do_list,
IDO_DO_GROUP_QTY );
9533:  if ( error_code < 0 )
9534:  {
9535:    state = TEST_STATE_QUIT_ERROR;
9536:  }
9537:
9538:  if ( true == Character_Get( NULL ) ) state = TEST_STATE_QUIT_USER;
9539:
9540:  if ( TEST_STATE_ACTIVE == state )
9541:  {
9542:    channel++;
9543:    sprintf( msg, "." );
9544:    Print_Multiple( msg, out );
9545:    if ( channel >= IDO_DO_QTY ) state = TEST_STATE_COMPLETE;
9546:  }
9547: }
9548:
9549:
9550: {
9551:  size_t index;
9552:  for ( index = 0; index < IDO_DO_GROUP_QTY; index++ )
9553:  {
9554:    IDO_DO_Group_Set( board_id, index, 0x00 );
9555:  }
9556: #if defined( __MSDOS__ )
9557:  delay( wait_ms );
9558: #endif
9559: }
9560:
9561:
9562:  sprintf( msg, "\n" );
9563:  Print_Multiple( msg, out );
9564:  switch( state )
9565:  {
9566:    case TEST_STATE_COMPLETE:
9567:      sprintf( msg, "PASS\n" );
9568:      Print_Multiple( msg, out );
9569:      error_code = SUCCESS;
9570:      break;
9571:    case TEST_STATE_QUIT_ERROR:
9572:      sprintf( msg, "FAIL at channel %d\n", (int) channel );
```

```
9573:     Print_Multiple( msg, out );
9574:     error_code = EC_TEST_FAIL;
9575:
9576:     for( group = 0; group < IDO_DO_GROUP_QTY; group++ )
9577:     {
9578:         IDO_DO_Group_Get( board_id, group, &(dog[group]) );
9579:     }
9580:     Print_Byt_List( "DO:",
9581:                     1,
9582:                     mode_string,
9583:                     IDO_DO_GROUP_QTY,
9584:                     dog,
9585:                     stdout
9586:                     );
9587:
9588:     if ( out != stdout )
9589:     {
9590:         Print_Byt_List( "DO:",
9591:                         1,
9592:                         mode_string,
9593:                         IDO_DO_GROUP_QTY,
9594:                         dog,
9595:                         out
9596:                         );
9597:     }
9598:     break;
9599: case TEST_STATE_QUIT_USER:
9600:     sprintf( msg, "CANCEL BY USER at channel %d\n", (int) channel );
9601:     Print_Multiple( msg, out );
9602:     error_code = EC_TEST_FAIL;
9603:     break;
9604: default:
9605:     break;
9606: }
9607: if ( out != stdout ) fclose( out );
9608: return error_code;
9609: }
9610:
9611: #endif
9612:
9613: static int CMD__IDO48_DOUT_Test_Alternate( int board_id, int argc, char * argv[ ] )
9614: {
9615:     TEST_STATE_ENUM state;
9616:     size_t group;
9617:     #if defined( __MSDOS__ )
9618:     size_t delay_normal_ms;
9619:     size_t delay_inverted_ms;
9620:     #endif
9621:     uint8_t pattern;
9622:     uint8_t dog[IDO_DO_GROUP_QTY];
9623:     char * mode_string[] = { "binary" };
9624:
9625:     state = TEST_STATE_INIT;
9626:     #if defined( __MSDOS__ )
9627:     delay_normal_ms = 100;
9628:     delay_inverted_ms = 20;
9629:     #endif
9630:     pattern = 0x55;
9631:
9632:     if ( argc > 0 )
9633:     {
9634:         pattern = (uint8_t) strtol( argv[0], NULL, 0 );
9635:     }
9636:
```

```
9640: if ( argc > 1 )
9641: {
9642:     if ( 0 == strcasecmp( "time", argv[1] ) )
9643:     {
9644:         if ( argc > 2 )
9645:         {
9646: #if defined( __MSDOS__ )
9647:             delay_normal_ms = (size_t) strtol( argv[2], NULL, 0 );
9648: #endif
9649:         }
9650:     else if ( argc > 3 )
9651:     {
9652: #if defined( __MSDOS__ )
9653:             delay_inverted_ms = (size_t) strtol( argv[3], NULL, 0 );
9654: #endif
9655:         }
9656:     }
9657: }
9658:
9659:
9660: for( group = 0; group < IDO_DO_GROUP_QTY; group++ )
9661: {
9662:     IDO_DO_Group_Set( board_id, group, 0x00 );
9663:     dog[group] = 0x00;
9664: }
9665:
9666: state = TEST_STATE_ACTIVE;
9667: printf( "IDO48 Alternating sequence testing (press any key to quit):" );
9668: while ( TEST_STATE_ACTIVE == state )
9669: {
9670:     for( group = 0; group < IDO_DO_GROUP_QTY; group++ )
9671:     {
9672:         IDO_DO_Group_Set( board_id, group, pattern );
9673:         dog[group] = pattern;
9674:     }
9675:
9676:     for( group = 0; group < IDO_DO_GROUP_QTY; group++ )
9677:     {
9678:         IDO_DO_Group_Get( board_id, group, &(dog[group]) );
9679:     }
9680:     Print_Byte_List( "DO:",
9681:                     1,
9682:                     mode_string,
9683:                     IDO_DO_GROUP_QTY,
9684:                     dog,
9685:                     stdout
9686:                     );
9687: #if defined( __MSDOS__ )
9688:     delay( delay_normal_ms );
9689: #endif
9690:
9691:     for( group = 0; group < IDO_DO_GROUP_QTY; group++ )
9692:     {
9693:         IDO_DO_Group_Set( board_id, group, ~pattern );
9694:         dog[group] = ~pattern;
9695:     }
9696:
9697:     for( group = 0; group < IDO_DO_GROUP_QTY; group++ )
9698:     {
9699:         IDO_DO_Group_Get( board_id, group, &(dog[group]) );
9700:     }
9701:     Print_Byte_List( "DO:",
9702:                     1,
9703:                     mode_string,
```

```
9704:     IDO_DO_GROUP_QTY,
9705:     dog,
9706:     stdout
9707:     );
9708: #if defined( __MSDOS__ )
9709:   delay( delay_inverted_ms );
9710: #endif
9711:   if ( true == Character_Get( NULL ) ) state = TEST_STATE_QUIT_USER;
9712: }
9713:
9714:
9715: for( group = 0; group < IDO_DO_GROUP_QTY; group++ )
9716: {
9717:   IDO_DO_Group_Set( board_id, group, 0x00 );
9718: }
9719: return SUCCESS;
9720: }
9721:
9725: static int CMD_IDO48_DOUT_Test_One_Hot( int board_id, int argc, char * argv[] )
9726: {
9727:   TEST_STATE_ENUM state;
9728:   size_t group;
9729:   size_t channel;
9730: #if defined( __MSDOS__ )
9731:   size_t delay_on_ms;
9732:   size_t delay_off_ms;
9733: #endif
9734:   uint8_t dog[IDO_DO_GROUP_QTY];
9735:   char * mode_string[] = { "binary" };
9736:
9737:   state = TEST_STATE_INIT;
9738: #if defined( __MSDOS__ )
9739:   delay_on_ms = 100;
9740:   delay_off_ms = 20;
9741: #endif
9742:   if ( argc > 0 )
9743:   {
9744: #if defined( __MSDOS__ )
9745:   delay_on_ms = (size_t) strtol( argv[0], NULL, 0 );
9746: #endif
9747:   }
9748:   else if ( argc > 1 )
9749:   {
9750: #if defined( __MSDOS__ )
9751:   delay_off_ms = (size_t) strtol( argv[1], NULL, 0 );
9752: #endif
9753:   }
9754:
9755:
9756: for( group = 0; group < IDO_DO_GROUP_QTY; group++ )
9757: {
9758:   dog[group] = 0x00;
9759:   IDO_DO_Group_Set( board_id, group, dog[group] );
9760: }
9761:
9762: state = TEST_STATE_ACTIVE;
9763: printf( "IDO48 One-hot sequence testing (press any key to quit):" );
9764: channel = 0;
9765: while ( TEST_STATE_ACTIVE == state )
9766: {
9767:   IDO_DO_Channel_Set( board_id, channel, true );
9768:
9769: for( group = 0; group < IDO_DO_GROUP_QTY; group++ )
9770: {
```

```

9771:     IDO_DO_Group_Get( board_id, group, &(dog[group]) );
9772: }
9773: Print_BytE_List( "DO:",
9774:     1,
9775:     mode_string,
9776:     IDO_DO_GROUP_QTY,
9777:     dog,
9778:     stdout
9779: );
9780: #if defined( __MSDOS__ )
9781:     delay( delay_on_ms );
9782: #endif
9783:
9784:     IDO_DO_Channel_Set( board_id, channel, false );
9785: #if defined( __MSDOS__ )
9786:     delay( delay_off_ms );
9787: #endif
9788:
9789:     for( group = 0; group < IDO_DO_GROUP_QTY; group++ )
9790:     {
9791:         IDO_DO_Group_Get( board_id, group, &(dog[group]) );
9792:     }
9793:     Print_BytE_List( "DO:",
9794:         1,
9795:         mode_string,
9796:         IDO_DO_GROUP_QTY,
9797:         dog,
9798:         stdout
9799:     );
9800:
9801:     if ( true == Character_Get( NULL ) ) state = TEST_STATE_QUIT_USER;
9802:     channel++;
9803:     if ( channel >= IDO_DO_QTY ) channel = 0;
9804: }
9805:
9806:
9807:     for( group = 0; group < IDO_DO_GROUP_QTY; group++ )
9808:     {
9809:         IDO_DO_Group_Set( board_id, group, 0x00 );
9810:     }
9811:     return SUCCESS;
9812: }
9813:
9814: static struct command_line_board cmd_ido48_test[] =
9815: {
9816:     { NULL, CMD__IDO48_DOUT_Test_One_Hot, "onehot", "toggling one at a time forever" },
9817:     { NULL, CMD__IDO48_DOUT_Test_Alternate, "alternate", "alternate pattern at all ports forever" },
9818: #if ( IDO_DO_GROUP_QTY == IDI_DIN_GROUP_QTY )
9819:     { NULL, CMD__IDO48_IDI48_Loopback, "loopback", "IDI48 & IDO48 Loopback test" },
9820: #endif
9821:     { NULL, NULL, NULL, NULL },
9822: };
9823:
9824:
9825:
9826: static int CMD__IDO48_Test( int board_id, int argc, char * argv[] )
9827: {
9828:     int error_code;
9829:     int index;
9830:     int argc_new;
9831:     char ** argv_new;
9832:     char * endptr;
9833:
9834:     if ( argc < 1 ) return -EC_NOT_FOUND;
9835:
```

```
9841: error_code = -EC_SYNTAX;
9842:
9843: strtol( argv[0], &endptr, 0 );
9844: if ( argv[0] != endptr )
9845: {
9846:     error_code = (* cmd_ido48_test[0].cmd_fnc )( board_id, argc, argv );
9847: }
9848: else
9849: {
9850:     index = 0;
9851:     while ( NULL != cmd_ido48_test[index].cmd_fnc )
9852:     {
9853:         if ( 0 == strcasecmp( cmd_ido48_test[index].name, argv[0] ) )
9854:         {
9855:             argv_new = &(argv[1]);
9856:             argc_new = argc - 1;
9857:             if ( 0 == argc_new ) argv_new = NULL;
9858:             error_code = (* cmd_ido48_test[index].cmd_fnc )( board_id, argc_new, argv_new );
9859:             break;
9860:         }
9861:         index++;
9862:     }
9863: }
9864: return error_code;
9865: }
9866:
9872: static int CMD__IDO48_DO_ID( int board_id, int argc, char * argv[] )
9873: {
9874:     uint16_t id;
9875:     (void) argc;
9876:     (void) argv;
9877:     IDO_DOUT_ID_Get( board_id, &id );
9878:     printf( "IDO48 DOUT ID: 0x%04X\n", id );
9879:     return SUCCESS;
9880: }
9886: static int CMD__IDO48_DO_All( int board_id, int argc, char * argv[] )
9887: {
9888:     int error_code;
9889:     size_t group;
9890:     uint8_t do_grp[6];
9891:
9892:
9893:
9894:     for ( group = 0; group < IDO_DO_GROUP_QTY; group++ )
9895:     {
9896:         error_code = IDO_DO_Group_Get( board_id, group, &(do_grp[group]) );
9897:     }
9898:
9899:     Print_Byte_List( "DO:" ,
9900:                     argc,
9901:                     argv,
9902:                     IDO_DO_GROUP_QTY,
9903:                     do_grp,
9904:                     stdout
9905:                     );
9906:     return SUCCESS;
9907: }
9908:
9914: static int CMD__IDO48_DO_Channel( int board_id, int argc, char * argv[] )
9915: {
9916:     int error_code;
9917:     int channel;
9918:     char message[8];
9919:     BOOL value;
```

```
9920:
9921: if ( argc < 1 ) return -EC_NOT_FOUND;
9922:
9923: channel = (int) strtol( argv[0], NULL, 0 );
9924:
9925: if ( argc < 2 )
9926: {
9927:     error_code = IDO_DO_Channel_Get( board_id, channel, &value );
9928:     message[0] = value ? '1' : '0';
9929:     message[1] = '\0';
9930:     printf( "DO%02d: %s\n", channel, message );
9931: }
9932: else
9933: {
9934:     value = String_To_Bool( argv[1] );
9935:     error_code = IDO_DO_Channel_Set( board_id, channel, value );
9936: }
9937: return SUCCESS;
9938: }
9939:
9940: static int CMD__IDO48_DO_Group( int board_id, int argc, char * argv[] )
9941: {
9942:     int error_code;
9943:     int group;
9944:     int group_start;
9945:     int group_end;
9946:
9947:     BOOL read_cycle;
9948:     uint8_t do_grp[IDO_DO_GROUP_QTY];
9949:
9950:
9951:
9952:     if ( argc < 1 )
9953:     {
9954:         group_start = 0;
9955:         group_end   = IDO_DO_GROUP_QTY - 1;
9956:         read_cycle  = true;
9957:     }
9958:     else if ( argc > 1 )
9959:     {
9960:         group_start = (int) strtol( argv[0], NULL, 0 );
9961:         if ( group_start >= IDO_DO_GROUP_QTY ) return -EC_CHANNEL;
9962:         group_end   = IDO_DO_GROUP_QTY - 1;
9963:         read_cycle  = false;
9964:     }
9965:     else
9966:     {
9967:         group_start = (int) strtol( argv[0], NULL, 0 );
9968:         if ( group_start >= IDO_DO_GROUP_QTY ) return -EC_CHANNEL;
9969:         group_end   = group_start + 1;
9970:         read_cycle  = true;
9971:     }
9972:
9973:     for ( group = group_start; group < group_end; group++ )
9974:     {
9975:         error_code = IDO_DO_Group_Get( board_id, group, &(do_grp[group]) );
9976:     }
9977:
9978:     printf( "DO_GROUP:" );
9979:     for ( group = group_start; group < group_end; group++ )
```

```
9989: {
9990:     printf( " 0x%02X", ((int) do_grp[group]) );
9991: }
9992: printf( "\n" );
9993: }
9994: else
9995: {
9996:     int index;
9997:     group_end = group_start;
9998:     group      = group_start;
9999:     for ( index = 1; index < argc; index++ )
10000:    {
10001:        do_grp[group] = (int) strtol( argv[index], NULL, 0 );
10002:        group++;
10003:        group_end++;
10004:        if ( group >= IDO_DO_GROUP_QTY ) break;
10005:    }
10006:
10007:    for ( group = group_start; group < group_end; group++ )
10008:    {
10009:        error_code = IDO_DO_Group_Set( board_id, group, do_grp[group] );
10010:    }
10011: }
10012: return SUCCESS;
10013: }
10014:
10017: static struct command_line_board cmd_ido48[] =
10018: {
10019: { NULL,      CMD__IDO48_DO_ID,   "id",      "params: none. Reports the DOUT board/component
ID." },
10020: { NULL,      CMD__IDO48_DO_Channel, "chan",    "params: <channel> <value>"           },
10021: { NULL,      CMD__IDO48_DO_Group,  "group",   "params: [<group_channel | all>]"       },
10022: { NULL,      CMD__IDO48_DO_All,    "all",     "gets all digital outputs in binary and hex"
},
10023: { cmd_ido48_test, CMD__IDO48_Test,  "test",    "semi-auto test functions"          },
10024: { NULL,      NULL,      NULL,      NULL           }
10025: };
10026:
10032: int Command_Line_IDO48( int board_id, int argc, char* argv[] )
10033: {
10034:     int error_code;
10035:     int index;
10036:     int argc_new;
10037:     char ** argv_new;
10038:     char * endptr;
10039:
10040:
10041:     if ( argc < 1 ) return -EC_NOT_FOUND;
10042:
10043:     if ( ID_IDO48 != board_id ) return -EC_BOARD_TYPE;
10044:
10045:     error_code = -EC_SYNTAX;
10046:
10047:
10048:     strtol( argv[0], &endptr, 0 );
10049:     if ( argv[0] != endptr )
10050:     {
10051:         error_code = (* cmd_ido48[0].cmd_fnc )( board_id, argc, argv );
10052:     }
10053:     else
10054:     {
10055:         index = 0;
10056:         while ( NULL != cmd_ido48[index].cmd_fnc )
10057:         {
```

```
10058:     if ( 0 == strcmpi( cmd_ido48[index].name, argv[0] ) )
10059:     {
10060:         argv_new = &(argv[1]);
10061:         argc_new = argc - 1;
10062:         if ( 0 == argc_new ) argv_new = NULL;
10063:         error_code = (* cmd_ido48[index].cmd_fnc )( board_id, argc_new, argv_new );
10064:         break;
10065:     }
10066:     index++;
10067: }
10068: }
10069: return error_code;
10070: }
10071:
10072:
10073:
10074:
10075:
10076:
10077:
10078:
10079:
10080:
10081:
10082:
10083: #define DIN_TEST_DEBUG_PRINT 1
10084:
10085:
10086:
10087:
10088:
10089:
10090: static BOOL CMD__IDI48_DIN_Helper_All_Values_False( uint8_t * dg, size_t group_qty )
10091: {
10092:     size_t group;
10093:
10094:     for ( group = 0; group < group_qty; group++ )
10095:     {
10096:         if ( 0x00 != dg[group] ) return false;
10097:     }
10098:     return true;
10099: }
10100:
10101:
10102:
10103:
10104:
10105:
10106: static int CMD__IDI48_DIN_Test_Value( int board_id, int argc, char * argv[] )
10107: {
10108:     int error_code;
10109:     TEST_STATE_ENUM state;
10110:     size_t group;
10111:     size_t bit;
10112:     size_t index;
10113:     uint8_t result;
10114:     uint8_t dig[IDI_DIN_GROUP_QTY];
10115:     uint8_t dig_prev[IDI_DIN_GROUP_QTY];
10116:     uint8_t group_change;
10117:     uint8_t mask;
10118:     BOOL din_cmplt[IDI_DIN_QTY];
10119:     FILE * out;
10120:     struct idi_dataset * dataset;
10121:     char * file_name;
10122:     char * msg;
10123:     char out_default[] = { "stdout" };
10124:     (void) argc;
10125:     (void) argv;
10126:
10127:     state = TEST_STATE_INIT;
```

```
10131:
10132: dataset = NULL;
10133: switch( board_id )
10134: {
10135:   case ID_IDI48:
10136:     dataset = ( struct idi_dataset * ) board_definition[board_id].dataset;
10137:     break;
10138:   default:
10139:     return -EC_BOARD_TYPE;
10140: }
10141:
10142:
10143: msg    = &(dataset->message[0]);
10144:
10145: out      = stdout;
10146: file_name = out_default;
10147: if ( argc > 0 )
10148: {
10149:   size_t wi;
10150:   char open_method[4];
10151:   wi = 0;
10152:   strcpy( open_method, "wt" );
10153:   if ( ( 0 == strcmpi( "--a", argv[wi] ) ) || |
10154:       ( 0 == strcmpi( "--append", argv[wi] ) ) )
10155:   {
10156:     wi++;
10157:     strcpy( open_method, "at" );
10158:   }
10159:
10160:   if ( argc > wi )
10161:   {
10162:     out = fopen( argv[wi], open_method );
10163:     file_name = argv[wi];
10164:   }
10165: }
10166:
10167: for( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10168: {
10169:   IDI_DIN_Group_Get( board_id, group, &(dig[group]) );
10170:   dig_prev[group] = dig[group];
10171: }
10172: for ( bit = 0; bit < IDI_DIN_QTY; bit++ ) din_cmplt[bit] = false;
10173:
10174: sprintf( msg, "DIN_TEST_VALUE:" ); Print_Multiple( msg, out );
10175: while ( ( TEST_STATE_QUIT != state ) && ( TEST_STATE_COMPLETE != state ) )
10176: {
10177:   for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ ) IDI_DIN_Group_Get( board_id,
group, &(dig[group]) );
10178:   switch( state )
10179:   {
10180:     case TEST_STATE_INIT:
10181:       if ( true == CMD_IDI48_DIN_Helper_All_Values_False( dig, IDI_DIN_GROUP_QTY ) )
10182:       {
10183:         state = TEST_STATE_READY;
10184:         sprintf( msg, "READY:" ); Print_Multiple( msg, out );
10185:       }
10186:       break;
10187:     case TEST_STATE_READY:
10188:       sprintf( msg, " " ); Print_Multiple( msg, out );
10189:       state = TEST_STATE_ACTIVE;
10190:       break;
10191:     case TEST_STATE_ACTIVE:
10192:       for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10193:     {
```

```

10194:     group_change = dig[group] ^ dig_prev[group];
10195:     if ( 0x00 != group_change )
10196:     {
10197:         mask = 0x01;
10198:         for ( bit = 0; bit < IDI_DIN_GROUP_SIZE; bit++ )
10199:         {
10200:             index = bit + group * IDI_DIN_GROUP_SIZE;
10201:             if ( ( 0x00 != ( mask & group_change ) ) && ( false == din_cmplt[index] ) )
10202:             {
10203:                 din_cmplt[index] = true;
10204:                 sprintf( msg, "." ); Print_Multiple( msg, out );
10205:             }
10206:             mask = mask << 1;
10207:         }
10208:     }
10209:     result = true;
10210:     for ( bit = 0; bit < IDI_DIN_QTY; bit++ ) result = result & din_cmplt[bit];
10211:     if ( true == result ) state = TEST_STATE_COMPLETE;
10212:     break;
10213: case TEST_STATE_COMPLETE:
10214:     break;
10215: case TEST_STATE_QUIT:
10216:     break;
10217: default:
10218:     break;
10219: }
10220: if ( true == Character_Get( NULL ) )
10221: {
10222:     state = TEST_STATE_QUIT;
10223: }
10224: for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ ) dig_prev[group] = dig[group];
10225: }
10226: }
10227: {
10228:     char * msg = &(dataset->message[0]);
10229:     Time_Current_String( msg, MESSAGE_SIZE ); Print_Multiple( msg, out );
10230:     sprintf( msg, ":" ); Print_Multiple( msg, out );
10231: }
10232: }
10233: if ( TEST_STATE_QUIT == state )
10234: {
10235:     sprintf( msg, "FAIL" ); error_code = EC_TEST_FAIL;
10236: }
10237: else
10238: {
10239:     sprintf( msg, "PASS" ); error_code = SUCCESS;
10240: }
10241: }
10242: Print_Multiple( msg, out );
10243: sprintf( msg, "\n" );
10244: Print_Multiple( msg, out );
10245: if ( out != stdout ) fclose( out );
10246: return error_code;
10247: }
10248: }
10249: static int CMD_IDI48_DIN_Test_RE( int board_id, int argc, char * argv[] )
10250: {
10251:     int error_code;
10252:     TEST_STATE_ENUM state;
10253:     size_t group;
10254:     size_t bit;
10255:     size_t index;
10256:     uint8_t result;
10257:     uint8_t dig[IDI_DIN_GROUP_QTY];

```

```
10261: uint8_t pending[IDI_DIN_GROUP_QTY];
10262: uint8_t mask;
10263: BOOL din_cmplt[IDI_DIN_QTY];
10264: #if defined( DIN_TEST_DEBUG_PRINT )
10265: char message[64];
10266: int cp;
10267: int state_detect;
10268: #endif
10269: (void) argc;
10270: (void) argv;
10271:
10272: state = TEST_STATE_INIT;
10273:
10274: for( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10275: {
10276:     IDI_DIN_Group_Get( board_id, group, &(dig[group]) );
10277:     IO_Write_U8( board_id, __LINE__, IDI_EDGE_GROUP0 + group, 0xFF );
10278:     IO_Write_U8( board_id, __LINE__, IDI_CLEAR_GROUP0 + group, 0xFF );
10279:     IO_Read_U8( board_id, __LINE__, IDI_PEND_GROUP0 + group, &(pending[group]) );
10280: }
10281: for ( bit = 0; bit < IDI_DIN_QTY; bit++ ) din_cmplt[bit] = false;
10282:
10283: printf( "DIN_TEST_RISING_EDGE:" );
10284: for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10285: {
10286:     IO_Read_U8( board_id, __LINE__, IDI_PEND_GROUP0 + group, &(pending[group]) );
10287:     IDI_DIN_Group_Get( board_id, group, &(dig[group]) );
10288:     IO_Write_U8( board_id, __LINE__, IDI_INTR_BIT_GROUP0 + group, 0xFF );
10289: }
10290: while ( ( TEST_STATE_QUIT != state ) && ( TEST_STATE_COMPLETE != state ) )
10291: {
10292:     switch( state )
10293:     {
10294:         case TEST_STATE_INIT:
10295:             if ( true == CMD_IDI48_DIN_Helper_All_Values_False( dig, IDI_DIN_GROUP_QTY ) )
10296:             {
10297:                 state = TEST_STATE_READY;
10298:                 printf( " READY" );
10299:             }
10300:             for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10301:             {
10302:                 IDI_DIN_Group_Get( board_id, group, &(dig[group]) );
10303:                 IO_Write_U8( board_id, __LINE__, IDI_CLEAR_GROUP0 + group, 0xFF );
10304:             }
10305:             break;
10306:         case TEST_STATE_READY:
10307:             #if defined( DIN_TEST_DEBUG_PRINT )
10308:                 cp = 0;
10309:                 strcpy( message, "" );
10310:                 state_detect = 2;
10311:                 printf( "\nPENDING:" );
10312:                 for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10313:                 {
10314:                     printf( " 0x%02X", pending[group] );
10315:                 }
10316:             #else
10317:                 printf( " " );
10318:             #endif
10319:             state = TEST_STATE_ACTIVE;
10320:             break;
10321:         case TEST_STATE_ACTIVE:
10322:             for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10323:             {
10324:                 if ( 0x00 != pending[group] )
```

```

10325:     {
10326:         mask = 0x01;
10327:         for ( bit = 0; bit < IDI_DIN_GROUP_SIZE; bit++ )
10328:         {
10329:             index = bit + group * IDI_DIN_GROUP_SIZE;
10330:             if ( ( 0x00 != (mask & pending[group]) ) && ( 0x00 != (dig[group] & mask) ) )
10331:             {
10332:                 if ( false == din_cmplt[index] )
10333:                 {
10334:                     din_cmplt[index] = true;
10335:                 }
10336:             #if defined( DIN_TEST_DEBUG_PRINT )
10337:             cp += sprintf( &(message[cp]), " [%2d]", (int) index );
10338:             state_detect = 1;
10339:         #else
10340:             printf( " ." );
10341:         #endif
10342:     }
10343:     }
10344:     mask = mask << 1;
10345:   }
10346:   IO_Write_U8( board_id, __LINE__, IDI_CLEAR_GROUP0 + group, 0xFF );
10347: }
10348: }
10349: #if defined( DIN_TEST_DEBUG_PRINT )
10350: if ( state_detect < 3 )
10351: {
10352:     printf( "\nPENDING:" );
10353:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10354:     {
10355:         printf( " 0x%02X", pending[group] );
10356:     }
10357:     printf( "    " );
10358:     printf( "%s", message );
10359:     cp = 0;
10360:     state_detect++;
10361: }
10362: strcpy( message, "" );
10363: #endif
10364:     result = true;
10365:     for ( bit = 0; bit < IDI_DIN_QTY; bit++ ) result = result & din_cmplt[bit];
10366:     if ( true == result ) state = TEST_STATE_COMPLETE;
10367:
10368:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10369:     {
10370:         IO_Read_U8( board_id, __LINE__, IDI_PEND_GROUP0 + group, &(pending[group]) );
10371:         IDI_DIN_Group_Get( board_id, group, &(dig[group]) );
10372:     }
10373:     break;
10374:     case TEST_STATE_COMPLETE:
10375:     #if defined( DIN_TEST_DEBUG_PRINT )
10376:         strcpy( message, "" );
10377:         printf( "\nPENDING:" );
10378:         for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10379:         {
10380:             printf( " 0x%02X", pending[group] );
10381:         }
10382:         printf( "    " );
10383:         printf( "\n" );
10384:     #endif
10385:     break;
10386:     case TEST_STATE_QUIT:
10387:     break;
10388:     default:

```

```
10389:     break;
10390: }
10391: if ( true == Character_Get( NULL ) )
10392: {
10393:     state = TEST_STATE_QUIT;
10394: }
10395: }
10396:
10397:
10398: for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10399: {
10400:     IO_Write_U8( board_id, __LINE__, IDI_INTR_BIT_GROUP0 + group, 0x00 );
10401: }
10402:
10403: if ( TEST_STATE_QUIT == state )
10404: {
10405:     printf( " FAIL" ); error_code = EC_TEST_FAIL;
10406: }
10407: else
10408: {
10409:     printf( " PASS" ); error_code = SUCCESS;
10410: }
10411:
10412: printf( "\n" );
10413:
10414: return error_code;
10415: }
10416:
10420: static int CMD_IDI48_DIN_Test_FE( int board_id, int argc, char * argv[] )
10421: {
10422:     int error_code;
10423:     TEST_STATE_ENUM state;
10424:     size_t group;
10425:     size_t bit;
10426:     size_t index;
10427:     uint8_t result;
10428:     uint8_t dig[IDI_DIN_GROUP_QTY];
10429:     uint8_t pending[IDI_DIN_GROUP_QTY];
10430:     uint8_t mask;
10431:     BOOL din_cmplt[IDI_DIN_QTY];
10432: #if defined( DIN_TEST_DEBUG_PRINT )
10433:     char message[64];
10434:     int cp;
10435:     int state_detect;
10436: #endif
10437:     (void) argc;
10438:     (void) argv;
10439:
10440:     state = TEST_STATE_INIT;
10441:
10442:     for( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10443:     {
10444:         IDI_DIN_Group_Get( board_id, group, &(dig[group]) );
10445:         IO_Write_U8( board_id, __LINE__, IDI_EDGE_GROUP0 + group, 0x00 );
10446:         IO_Write_U8( board_id, __LINE__, IDI_CLEAR_GROUP0 + group, 0xFF );
10447:         IO_Read_U8( board_id, __LINE__, IDI_PEND_GROUP0 + group, &(pending[group]) );
10448:     }
10449:     for ( bit = 0; bit < IDI_DIN_QTY; bit++ ) din_cmplt[bit] = false;
10450:
10451:     printf( "DIN_TEST_FALLING_EDGE:" );
10452:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10453:     {
10454:         IO_Read_U8( board_id, __LINE__, IDI_PEND_GROUP0 + group, &(pending[group]) );
10455:         IDI_DIN_Group_Get( board_id, group, &(dig[group]) );
```

```

10456:     IO_Write_U8( board_id, __LINE__, IDI_INTR_BIT_GROUP0 + group, 0xFF );
10457: }
10458: while ( ( TEST_STATE_QUIT != state ) && ( TEST_STATE_COMPLETE != state ) )
10459: {
10460:     switch( state )
10461:     {
10462:         case TEST_STATE_INIT:
10463:             if ( true == CMD_IDI48_DIN_Helper_All_Values_False( dig, IDI_DIN_GROUP_QTY ) )
10464:             {
10465:                 state = TEST_STATE_READY;
10466:                 printf( " READY" );
10467:             }
10468:             for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10469:             {
10470:                 IDI_DIN_Group_Set( board_id, group, &(dig[group]) );
10471:                 IO_Write_U8( board_id, __LINE__, IDI_CLEAR_GROUP0 + group, 0xFF );
10472:             }
10473:             break;
10474:         case TEST_STATE_READY:
10475: #if defined( DIN_TEST_DEBUG_PRINT )
10476:             cp = 0;
10477:             strcpy( message, "" );
10478:             state_detect = 2;
10479:             printf( "\nPENDING:" );
10480:             for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10481:             {
10482:                 printf( " 0x%02X", pending[group] );
10483:             }
10484: #else
10485:             printf( " " );
10486: #endif
10487:             state = TEST_STATE_ACTIVE;
10488:             break;
10489:         case TEST_STATE_ACTIVE:
10490:             for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10491:             {
10492:                 if ( 0x00 != pending[group] )
10493:                 {
10494:                     mask = 0x01;
10495:                     for ( bit = 0; bit < IDI_DIN_GROUP_SIZE; bit++ )
10496:                     {
10497:                         index = bit + group * IDI_DIN_GROUP_SIZE;
10498:                         if ( ( 0x00 != (mask & pending[group]) ) && ( 0x00 == (dig[group] & mask) ) )
10499:                         {
10500:                             if ( false == din_cmplt[index] )
10501:                             {
10502:                                 din_cmplt[index] = true;
10503:                             }
10504: #if defined( DIN_TEST_DEBUG_PRINT )
10505: cp += sprintf( &(message[cp]), " [%2d]", (int) index );
10506: state_detect = 1;
10507: #else
10508:             printf( ".." );
10509: #endif
10510:             }
10511:             }
10512:             mask = mask << 1;
10513:         }
10514:         IO_Write_U8( board_id, __LINE__, IDI_CLEAR_GROUP0 + group, 0xFF );
10515:     }
10516: }
10517: #if defined( DIN_TEST_DEBUG_PRINT )
10518: if ( state_detect < 3 )
10519: {

```

```
10520: printf( "\nPENDING:" );
10521: for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10522: {
10523:   printf( " 0x%02X", pending[group] );
10524: }
10525: printf( "    " );
10526: printf( "%s", message );
10527: cp = 0;
10528: state_detect++;
10529: }
10530: strcpy( message, "" );
10531: #endif
10532:   result = true;
10533:   for ( bit = 0; bit < IDI_DIN_QTY; bit++ ) result = result & din_cmplt[bit];
10534:   if ( true == result ) state = TEST_STATE_COMPLETE;
10535:
10536:   for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10537:   {
10538:     IO_Read_U8( board_id, __LINE__, IDI_PEND_GROUP0 + group, &(pending[group]) );
10539:     IDI_DIN_Group_Get( board_id, group, &(dig[group]) );
10540:   }
10541:   break;
10542: case TEST_STATE_COMPLETE:
10543: #if defined( DIN_TEST_DEBUG_PRINT )
10544:   strcpy( message, "" );
10545:   printf( "\nPENDING:" );
10546:   for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10547:   {
10548:     printf( " 0x%02X", pending[group] );
10549:   }
10550:   printf( "    " );
10551:   printf( "\n" );
10552: #endif
10553:   break;
10554: case TEST_STATE_QUIT:
10555:   break;
10556: default:
10557:   break;
10558: }
10559: if ( true == Character_Get( NULL ) )
10560: {
10561:   state = TEST_STATE_QUIT;
10562: }
10563: }
10564:
10565:
10566: for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10567: {
10568:   IO_Write_U8( board_id, __LINE__, IDI_INTR_BIT_GROUP0 + group, 0x00 );
10569: }
10570:
10571: if ( TEST_STATE_QUIT == state )
10572: {
10573:   printf( " FAIL" ); error_code = EC_TEST_FAIL;
10574: }
10575: else
10576: {
10577:   printf( " PASS" ); error_code = SUCCESS;
10578: }
10579:
10580: printf( "\n" );
10581:
10582: return error_code;
10583: }
```

```

10584:
10585:
10586:
10587: #if defined( __MSDOS__ )
10588:
10589: static irqreturn_t CMD_IDI48_DIN_Test_Interrupt_Handler( int irq, void * dev_id,
10590: struct pt_regs * regs )
10591: {
10592:     size_t group;
10593:     uint8_t interrupt_status;
10594:     uint8_t mask_group;
10595:     uint8_t pending;
10596:     uint8_t bank_backup;
10597:     uint8_t bank_previous_backup;
10598:     int board_id;
10599:     struct idi_dataset * idi_dataset;
10600:     (void) irq;
10601:     (void) dev_id;
10602:     (void) regs;
10603:
10604:
10605:
10606:
10607:
10608:
10609:
10610:     board_id = *((int *) dev_id);
10611:     switch( board_id )
10612:     {
10613:         case ID_IDI48:
10614:             idi_dataset = ( struct idi_dataset * ) board_definition[board_id].dataset;
10615:             break;
10616:         default:
10617:             goto CMD_IDI48_DIN_Test_Interrupt_Handler_Completed;
10618:     }
10619:
10620:
10621:     bank_previous_backup = idi_dataset->bank_previous;
10622:     IO_Read_U8( board_id, __LINE__, IDI_BANK, &bank_backup );
10623:
10624:     IO_Read_U8( board_id, __LINE__, IDI_INTR_BY_GROUP, &interrupt_status );
10625:     mask_group = 0x01;
10626:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10627:     {
10628:         if ( 0 != ( mask_group & interrupt_status ) )
10629:         {
10630:             IO_Read_U8( board_id, __LINE__, IDI_PEND_GROUP0 + group, &pending );
10631:             idi_dataset->isr_pending_list[group] = pending;
10632:             IO_Write_U8( board_id, __LINE__, IDI_CLEAR_GROUP0 + group, pending );
10633:         }
10634:         else
10635:         {
10636:             idi_dataset->isr_pending_list[group] = 0;
10637:         }
10638:         mask_group = mask_group << 1;
10639:     }
10640:
10641:     IO_Write_U8( board_id, __LINE__, IDI_BANK, bank_backup );
10642:     idi_dataset->bank_previous = bank_previous_backup;
10643:     CMD_IDI48_DIN_Test_Interrupt_Handler_Completed:
10644:     idi_dataset->irq_count++;
10645:     return IRQ_HANDLED;
10646: }
10647: #endif
10648:
10649:
10650: static int CMD_IDI48_DIN_Test_Interrupt( int board_id, int argc, char * argv[] )
10651: {
10652:     struct idi_dataset * idi_dataset;
10653:
```

```
10659: int error_code;
10660: TEST_STATE_ENUM state;
10661: size_t group;
10662: size_t bit;
10663: size_t index;
10664: size_t irq_count_local;
10665: uint8_t result;
10666: uint8_t irq_pending_local;
10667: uint8_t dig[IDI_DIN_GROUP_QTY];
10668: uint8_t pending[IDI_DIN_GROUP_QTY];
10669: uint8_t mask;
10670: BOOL io_report_backup, io_simulate_backup;
10671: BOOL din_cmplt[IDI_DIN_QTY];
10672: #if defined( DIN_TEST_DEBUG_PRINT )
10673: char message[64];
10674: int cp;
10675: int state_detect;
10676: #endif
10677: #if defined( __MSDOS__ )
10678: unsigned int interrupt_number;
10679: switch( board_id )
10680: {
10681: case ID_IDI48:
10682: idi_dataset = ( struct idi_dataset * ) board_definition[board_id].dataset;
10683: break;
10684: default:
10685: return -EC_BOARD;
10686: }
10687:
10688: if ( argc > 0 )
10689: {
10690: interrupt_number = (unsigned int) strtol( argv[0], NULL, 0 );
10691: }
10692: else
10693: {
10694: interrupt_number = idi_dataset->irq_number;
10695: }
10696: #else
10697:
10698: (void) argc;
10699: (void) argv;
10700:
10701: switch( board_id )
10702: {
10703: case ID_IDI48:
10704: idi_dataset = ( struct idi_dataset * ) board_definition[board_id].dataset;
10705: break;
10706: default:
10707: return -EC_BOARD;
10708: }
10709: #endif
10710:
10711: IO_Write_U8( board_id, __LINE__, IDI_SPI_CONFIG, 0 );
10712: io_report_backup = idi_dataset->io_report;
10713: io_simulate_backup = idi_dataset->io_simulate;
10714:
10715: idi_dataset->irq_count_previous = 0;
10716: idi_dataset->irq_count = 0;
10717:
10718: state = TEST_STATE_INIT;
10719: irq_pending_local = 0;
10720: irq_count_local = 0;
10721:
10722:
```

```

10723: for( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10724: {
10725:   IDI_DIN_Group_Get( board_id, group, &(dig[group]) );
10726:   IO_Write_U8( board_id, __LINE__, IDI_EDGE_GROUP0 + group, 0xFF );
10727:   IO_Write_U8( board_id, __LINE__, IDI_CLEAR_GROUP0 + group, 0xFF );
10728:   IO_Read_U8( board_id, __LINE__, IDI_PEND_GROUP0 + group, &(pending[group]) );
10729: }
10730: for ( bit = 0; bit < IDI_DIN_QTY; bit++ ) din_cmplt[bit] = false;
10731:
10732: printf( "DIN_INTERRUPT:" );
10733: for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10734: {
10735:   IO_Read_U8( board_id, __LINE__, IDI_PEND_GROUP0 + group, &(pending[group]) );
10736:   IDI_DIN_Group_Get( board_id, group, &(dig[group]) );
10737: }
10738:
10739: #if defined( __MSDOS__ )
10740:
10741:   error_code = IOKern_DOS_IRQ_Request( interrupt_number,
10742:                                         CMD_IDI48_DIN_Test_Interrupt_Handler,
10743:                                         (void *) &board_id
10744:                                       );
10745:   if ( SUCCESS != error_code )
10746:   {
10747:     state = TEST_STATE_QUIT;
10748:     goto CMD_IDI48_DIN_Test_Interrupt_Terminate;
10749:   }
10750: #else
10751:   state = TEST_STATE_QUIT;
10752:   error_code = -EC_INTERRUPT_UNAVAILABLE;
10753:   goto CMD_IDI48_DIN_Test_Interrupt_Terminate;
10754: #endif
10755:
10756:
10757: for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10758: {
10759:   IO_Write_U8( board_id, __LINE__, IDI_INTR_BIT_GROUP0 + group, 0xFF );
10760: }
10761:
10762: idi_dataset->irq_handler_active = true;
10763:
10764: while ( ( TEST_STATE_QUIT != state ) && ( TEST_STATE_COMPLETE != state ) )
10765: {
10766:
10767:   switch( state )
10768:   {
10769:     case TEST_STATE_INIT:
10770:       for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10771:       {
10772:         IDI_DIN_Group_Get( board_id, group, &(dig[group]) );
10773:       }
10774:       if ( true == CMD_IDI48_DIN_Helper_All_Values_False( dig, IDI_DIN_GROUP_QTY ) )
10775:       {
10776:         state = TEST_STATE_READY;
10777:         printf( " READY" );
10778:       }
10779:       for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ ) IO_Write_U8( board_id,
10780: __LINE__, IDI_CLEAR_GROUP0 + group, 0xFF );
10781:       break;
10782:     case TEST_STATE_READY:
10783:       #if defined( DIN_TEST_DEBUG_PRINT )
10784:         cp = 0;
10785:         strcpy( message, "" );
10786:         state_detect = 2;

```

```
10786:     printf( "\nPENDING:" );
10787:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10788:     {
10789:         printf( " 0x%02X", pending[group] );
10790:     }
10791: #else
10792:     printf( "  " );
10793: #endif
10794:     state = TEST_STATE_ACTIVE;
10795:     break;
10796: case TEST_STATE_ACTIVE:
10797:
10798:     if ( true == Character_Get( NULL ) ) state = TEST_STATE_QUIT;
10799:
10800: #if defined( __MSDOS__ )
10801:     disable();
10802: #endif
10803:     irq_count_local = idi_dataset->irq_count;
10804:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10805:     {
10806:         pending[group] = idi_dataset->isr_pending_list[group];
10807:         if ( 0 != pending[group] ) irq_pending_local++;
10808:     }
10809: #if defined( __MSDOS__ )
10810:     enable();
10811: #endif
10812:
10813:     if ( irq_count_local != idi_dataset->irq_count_previous )
10814:     {
10815:         if ( irq_count_local > idi_dataset->irq_count_previous )
10816:         {
10817:             index = irq_count_local - idi_dataset->irq_count_previous;
10818:         }
10819:     else
10820:     {
10821:         index = UINT_MAX - idi_dataset->irq_count_previous;
10822:         index = index + irq_count_local + 1;
10823:     }
10824:     idi_dataset->irq_count_previous = irq_count_local;
10825:     irq_count_local = index;
10826: }
10827: else
10828: {
10829:     irq_count_local = 0;
10830:     irq_pending_local = 0;
10831: }
10832:
10833:     if ( irq_count_local > 0 )
10834:     {
10835: #if defined( DIN_TEST_DEBUG_PRINT )
10836: cp += sprintf( &(message[cp]), " %2d", (int) irq_count_local );
10837: #endif
10838:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10839:     {
10840:         if ( 0x00 != pending[group] )
10841:         {
10842:             mask = 0x01;
10843:             for ( bit = 0; bit < IDI_DIN_GROUP_SIZE; bit++ )
10844:             {
10845:                 index = bit + group * IDI_DIN_GROUP_SIZE;
10846:                 if ( 0x00 != (mask & pending[group]) )
10847:                 {
10848:                     if ( false == din_cmplt[index] )
10849:                     {
```

```

10850:         din_cmplt[index] = true;
10851: #if defined( DIN_TEST_DEBUG_PRINT )
10852: cp += sprintf( &(message[cp]), " [%2d]", (int) index );
10853: state_detect = 1;
10854: #else
10855:         printf( "." );
10856: #endif
10857:         }
10858:         }
10859:         mask = mask << 1;
10860:         }
10861:         }
10862:         }
10863:         }
10864: #if defined( DIN_TEST_DEBUG_PRINT )
10865: if ( state_detect < 3 )
10866: {
10867:     printf( "\nPENDING:" );
10868:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10869:     {
10870:         printf( " 0x%02X", pending[group] );
10871:     }
10872:     printf( "    " );
10873:     printf( "%s", message );
10874:     cp = 0;
10875:     state_detect++;
10876: }
10877: strcpy( message, "" );
10878: #endif
10879:     result = true;
10880:     for ( bit = 0; bit < IDI_DIN_QTY; bit++ ) result = result & din_cmplt[bit];
10881:     if ( true == result ) state = TEST_STATE_COMPLETE;
10882:     break;
10883:     case TEST_STATE_COMPLETE:
10884: #if defined( DIN_TEST_DEBUG_PRINT )
10885:     strcpy( message, "" );
10886:     printf( "\nPENDING:" );
10887:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10888:     {
10889:         printf( " 0x%02X", pending[group] );
10890:     }
10891:     printf( "    " );
10892:     printf( "\n" );
10893: #endif
10894:     break;
10895:     case TEST_STATE_QUIT:
10896:     break;
10897:     default:
10898:     break;
10899: }
10900: if ( true == Character_Get( NULL ) )
10901: {
10902:     state = TEST_STATE_QUIT;
10903: }
10904: }
10905:
10906:
10907: for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
10908: {
10909:     IO_Write_U8( board_id, __LINE__, IDI_INTR_BIT_GROUP0 + group, 0x00 );
10910: }
10911:
10912: #if defined( __MSDOS__ )
10913: IOKern_DOS_IRQ_Free( idi_dataset->irq_number );

```

```
10914: idi_dataset->irq_handler_active = false;
10915: #endif
10916:
10917: CMD__IDI48_DIN_Test_Interrupt_Terminate:
10918:
10919: idi_dataset->io_report    = io_report_backup;
10920: idi_dataset->io_simulate = io_simulate_backup;
10921:
10922: if ( TEST_STATE_QUIT == state )
10923: {
10924:     printf( " FAIL" );   error_code = EC_TEST_FAIL;
10925: }
10926: else
10927: {
10928:     printf( " PASS" );  error_code = SUCCESS;
10929: }
10930:
10931: printf( "\n" );
10932:
10933: return error_code;
10934: }
10935:
10936: static struct command_line_board cmd_idi48_test[] =
10937: {
10938:     { NULL, CMD__IDI48_DIN_Test_Value, "value", "loop, until all have toggled" },
10939:     { NULL, CMD__IDI48_DIN_Test_RE,    "re",    "loop, rising-edge detect (led perspective)" },
10940:     { NULL, CMD__IDI48_DIN_Test_FE,    "fe",    "loop, falling-edge detect (led perspective)" },
10941:     { NULL, CMD__IDI48_DIN_Test_Interrupt, "interrupt", "loop, test interrupt on all inputs" },
10942:     { NULL, NULL,           NULL,      NULL       },
10943: };
10944:
10945: };
10946:
10947: static int CMD__IDI48_DIN_Test( int board_id, int argc, char * argv[] )
10948: {
10949:     int error_code;
10950:     int index;
10951:     int argc_new;
10952:     char ** argv_new;
10953:     char * endptr;
10954:
10955:
10956:
10957:
10958:
10959:     if ( argc < 1 ) return -EC_NOT_FOUND;
10960:
10961:     error_code = -EC_SYNTAX;
10962:
10963:
10964:     strtol( argv[0], &endptr, 0 );
10965:     if ( argv[0] != endptr )
10966:     {
10967:         error_code = (* cmd_idi48_test[0].cmd_fnc )( board_id, argc, argv );
10968:     }
10969:     else
10970:     {
10971:         index = 0;
10972:         while ( NULL != cmd_idi48_test[index].cmd_fnc )
10973:         {
10974:             if ( 0 == strcasecmp( cmd_idi48_test[index].name, argv[0] ) )
10975:             {
10976:                 argv_new = &(argv[1]);
10977:                 argc_new = argc - 1;
10978:                 if ( 0 == argc_new ) argv_new = NULL;
10979:                 error_code = (* cmd_idi48_test[index].cmd_fnc )( board_id, argc_new, argv_new );
```

```

10980:     break;
10981:   }
10982:   index++;
10983: }
10984: }
10985: return error_code;
10986: }
10987:
10988: struct port_list
10989: {
10990:   int address_start;
10991:   int address_end;
10992:   BOOL scan;
10993:   const char * name;
10994: };
10995:
10996: static const struct port_list lpm_lx800_port_list[] =
10997: {
10998:   { 0x000, 0x00F, 0, "8237DMA Controller #1" },
10999:   { 0x010, 0x01F, 1, "Free" },
11000:   { 0x020, 0x021, 0, "8259 PIC #1" },
11001:   { 0x022, 0x03F, 1, "Free" },
11002:   { 0x040, 0x043, 0, "8254 PIT" },
11003:   { 0x044, 0x04D, 1, "Free" },
11004:   { 0x04E, 0x04F, 0, "Reserved for on, 0xboard configuration" },
11005:   { 0x050, 0x05F, 1, "Free" },
11006:   { 0x060, 0x06F, 0, "8042 Keyboard / Mouse Controller" },
11007:   { 0x070, 0x07F, 0, "CMOS RAM, Clock / Calendar" },
11008:   { 0x080, 0x09F, 0, "DMA Page Registers" },
11009:   { 0x0A0, 0x0BF, 0, "8259 PIC #2" },
11010:   { 0x0C0, 0x0DF, 0, "8237 DMA Controller #2" },
11011:   { 0x0E0, 0x0EF, 1, "Free" },
11012:   { 0x0F0, 0x0F1, 0, "Math Co, 0xprocessor Control" },
11013:   { 0x0F2, 0x0F7, 1, "Free" },
11014:   { 0x0F8, 0x0FF, 0, "Math Co, 0xprocessor" },
11015:   { 0x100, 0x102, 0, "Video Controllers" },
11016:   { 0x103, 0x11F, 1, "Free" },
11017:   { 0x120, 0x12F, 0, "Digital I/O" },
11018:   { 0x130, 0x14F, 1, "Free" },
11019:   { 0x150, 0x150, 0, "Reserved for on, 0xboard configuration" },
11020:   { 0x151, 0x1CF, 1, "Free" },
11021:   { 0x1D0, 0x1DF, 0, "Legacy Watchdog (1D0, 0xEnabled; 1D8, 0x Pet)" },
11022:   { 0x1E8, 0x1EB, 0, "Reserved for on, 0xboard configuration" },
11023:   { 0x1EC, 0x1EC, 0, "Interrupt Status Register" },
11024:   { 0x1ED, 0x1ED, 0, "Status LED" },
11025:   { 0x1EE, 0x1EF, 0, "Watchdog Timer Control" },
11026:   { 0x1F0, 0x1FF, 0, "IDE Controller #1" },
11027:   { 0x200, 0x277, 1, "Free" },
11028:   { 0x278, 0x27F, 1, "Free (Option for LPT)" },
11029:   { 0x280, 0x2A7, 1, "Free" },
11030:   { 0x2A8, 0x2AF, 1, "Free (Option for on, 0xboard serial ports)" },
11031:   { 0x2B0, 0x2DF, 0, "Video Controllers" },
11032:   { 0x2E0, 0x2E7, 1, "Free" },
11033:   { 0x2E8, 0x2EF, 0, "COM4 - (Default)" },
11034:   { 0x2F0, 0x2F7, 1, "Free" },
11035:   { 0x2F8, 0x2FF, 0, "COM2 - (Default)" },
11036:   { 0x300, 0x377, 1, "Free" },
11037:   { 0x378, 0x37B, 0, "LPT (Default)" },
11038:   { 0x37C, 0x3A7, 1, "Free" },
11039:   { 0x3A8, 0x3AF, 1, "Free (Option for on, 0xboard serial ports)" },
11040:   { 0x3B0, 0x3BB, 0, "Video Controllers" },
11041:   { 0x3BC, 0x3BF, 1, "Free (Option for LPT)" },
11042:   { 0x3C0, 0x3DF, 0, "Video Controllers" },
11043:   { 0x3E0, 0x3E7, 1, "Free" }
}

```

```
11044: { 0x3E8, 0x3EF, 0, "COM3 - (Default)" },  
11045: { 0x3F0, 0x3F7, 1, "Free" } ,  
11046: { 0x000, 0x000, 0, NULL }  
11047: };  
11048:  
11049:  
11050:  
11051:  
11052: static int IDI48_DIN_ID_Port_Scan( const struct port_list * pl, size_t * row, int *  
port )  
11053: {  
11054:     size_t      row_index;  
11055:     int         pa;  
11056:     int         pi_start, pi_end;  
11057:     int         count;  
11058:     BOOL        not_done;  
11059:     uint16_t    id;  
11060:     uint8_t     lsb, msb;  
11061:  
11062: #if defined( __MSDOS__ )  
11063: #else  
11064:     return -EC_NOT_IMPLEMENTED;  
11065: #endif  
11066:  
11067:     not_done = true;  
11068:     row_index = 0;  
11069:     do  
11070:     {  
11071:         if ( ( 0 == pl[row_index].address_start ) && ( 0 == pl[row_index].address_end ) )  
11072:         {  
11073:             not_done = false;  
11074:         }  
11075:         else  
11076:         {  
11077:             printf ( "." );  
11078:             count = pl[row_index].address_end - pl[row_index].address_start + 1;  
11079:             if ( count > 1 )  
11080:             {  
11081:                 pi_start = pl[row_index].address_start ;  
11082:                 pi_end   = pl[row_index].address_end   + 1;  
11083:                 if ( 1 == ( pi_start & 1 ) )  
11084:                 {  
11085:                     pi_start++;  
11086:                     count--;  
11087:                 }  
11088:             }  
11089:             for ( pa = pi_start; pa < pi_end; pa = pa + 2 )  
11090:             {  
11091:                 if defined( __MSDOS__ )  
11092:                 lsb = importb( ( pa + 0 ) );  
11093:                 msb = importb( ( pa + 1 ) );  
11094: #endif  
11095:                 id = ( ((uint16_t) msb) << 8 ) | ((uint16_t) lsb);  
11096:                 if ( ID_DIN == id )  
11097:                 {  
11098:                     *row = row_index;  
11099:                     *port = pa;  
11100:                     return SUCCESS;  
11101:                 }  
11102:             }  
11103:         }  
11104:     }  
11105:     row_index++;  
11106: }
```

```
11112: } while ( true == not_done );
11113:
11114: return -EC_NOT_FOUND;
11115: }
11116:
11117:
11118:
11124: static int CMD_IDI48_DIN_ID( int board_id, int argc, char * argv[] )
11125: {
11126:     uint16_t id;
11127:     (void) argc;
11128:     (void) argv;
11129:
11130:     if ( argc > 0 )
11131:     {
11132:         if ( 0 == strcasecmp( "scan", argv[0] ) )
11133:         {
11134:             size_t row;
11135:             int port;
11136:             int error_code = SUCCESS;
11137:
11138:             printf ( "Scanning: " );
11139:             error_code = IDI48_DIN_ID_Port_Scan( lpm_lx800_port_list, &row, &port );
11140:             if ( SUCCESS == error_code )
11141:             {
11142:                 printf( "port found at row = %02d, id address = 0x%04X", (int) row, port );
11143:             }
11144:             else
11145:             {
11146:                 printf( "not found" );
11147:             }
11148:             printf( "\n" );
11149:
11150:         }
11151:     }
11152:     else
11153:     {
11154:         IDI_DIN_ID_Get( board_id, &id );
11155:         printf( "IDI48 DIN ID: 0x%04X\n", id );
11156:     }
11157:     return SUCCESS;
11158: }
11159:
11165: static int CMD_IDI48_DIN_All( int board_id, int argc, char * argv[] )
11166: {
11167:     int error_code;
11168:     int channel;
11169:
11170:     int cp;
11171:     enum { MODE_NONE = 0, MODE_BINARY = 1, MODE_HEX = 2, MODE_ALL = 3 } mode_out;
11172:     int group;
11173:     char message[64];
11174:     uint8_t din_grp[IDI_DIN_GROUP_QTY];
11175:     uint8_t mask;
11176:     (void) argc;
11177:     (void) argv;
11178:
11179:     mode_out = MODE_ALL;
11180:     if ( argc > 0 )
11181:     {
11182:         int index;
11183:         mode_out = MODE_NONE;
11184:         for ( index = 0; index < argc; index++ )
11185:     }
```

```
11186:     if      ( 0 == strcasecmp( "binary", argv[index] ) ) mode_out |= MODE_BINARY;
11187:     else if ( 0 == strcasecmp( "group", argv[index] ) ) mode_out |= MODE_HEX;
11188:     else if ( 0 == strcasecmp( "hex", argv[index] ) ) mode_out |= MODE_HEX;
11189:     else
11190:     }
11191: }
11192:
11193: cp      = 0;
11194: group  = 0;
11195: for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
11196: {
11197:     error_code = IDI_DIN_Group_Get( board_id, group, &(din_grp[group]) );
11198:     mask = 0x01;
11199:     for ( channel = 0; channel < 8; channel++ )
11200:     {
11201:         message[cp++] = !(din_grp[group] & mask) ? '1' : '0';
11202:         mask = mask << 1;
11203:     }
11204:     message[cp++] = ' ';
11205: }
11206: message[cp] = '\0';
11207: if ( MODE_BINARY == ( mode_out & MODE_BINARY ) )
11208: {
11209:     printf( "DIN: %s\n", message );
11210: }
11211: if ( MODE_HEX == ( mode_out & MODE_HEX ) )
11212: {
11213:     printf( "DIN:" );
11214:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ ) printf( " %02X", din_grp[group] );
11215:     printf( "\n" );
11216: }
11217:
11218: return SUCCESS;
11219: }
11220:
11221: static int CMD_IDI48_DIN_Channel( int board_id, int argc, char * argv[] )
11222: {
11223:     int error_code;
11224:     int channel;
11225:     char message[IDI_DIN_GROUP_QTY + 2];
11226:     BOOL value;
11227:
11228:     if ( argc < 1 ) return -EC_NOT_FOUND;
11229:
11230:     channel = (int) strtol( argv[0], NULL, 0 );
11231:     error_code = IDI_DIN_Channel_Get( board_id, channel, &value );
11232:     message[0] = value ? '1' : '0';
11233:     message[1] = '\0';
11234:
11235:     printf( "DIN%02d: %s\n", channel, message );
11236:     return SUCCESS;
11237: }
11238:
11239: static int CMD_IDI48_DIN_Group( int board_id, int argc, char * argv[] )
11240: {
11241:     int error_code;
11242:
11243:     int group;
11244:     int group_count;
11245:     BOOL do_all;
11246:     uint8_t din_grp[IDI_DIN_GROUP_QTY];
11247:
11248:     if ( argc < 1 ) do_all = true;
```

```

11259: else do_all = false;
11260:
11261: if ( 0 == strcasecmp( "all", argv[0] ) ) do_all = true;
11262:
11263: if ( do_all )
11264: {
11265:     for ( group = 0; group < IDI_DIN_GROUP_QTY; group++ )
11266:     {
11267:         error_code = IDI_DIN_Group_Get( board_id, group, &(din_grp[group]) );
11268:     }
11269:     group_count = IDI_DIN_GROUP_QTY;
11270: }
11271: else
11272: {
11273:     group = (int) strtol( argv[0], NULL, 0 );
11274:     error_code = IDI_DIN_Group_Get( board_id, group, &(din_grp[0]) );
11275:     group_count = 1;
11276: }
11277:
11278: printf( "DIN_GROUP:" );
11279: for ( group = 0; group < group_count; group++ )
11280: {
11281:     printf( " 0x%02X", ((int) din_grp[group]) );
11282: }
11283: printf( "\n" );
11284: return SUCCESS;
11285: }
11286:
11287: static struct command_line_board cmd_idi48[] =
11288: {
11289:     { NULL,      CMD_IDI48_DIN_ID,    "id",      "params: none. Reports the DIN board/component ID." },
11290:     { NULL,      CMD_IDI48_DIN_Channel, "chan",    "params: <channel>" },
11291:     { NULL,      CMD_IDI48_DIN_Group,   "group",   "params: [<group_channel | all>]" },
11292:     { NULL,      CMD_IDI48_DIN_All,    "all",     "reports all digital inputs in binary and hex" },
11293:     { cmd_idi48_test, CMD_IDI48_DIN_Test, "test",    "semi-auto test functions" },
11294:     { NULL,      NULL,        NULL,     NULL }
11295: };
11296:
11297:
11298:
11299:
11300: int Command_Line_IDI48( int board_id, int argc, char* argv[] )
11301: {
11302:     int error_code;
11303:     int index;
11304:     int argc_new;
11305:     char ** argv_new;
11306:     char * endptr;
11307:
11308:
11309:
11310:
11311:
11312:
11313:
11314:     if ( argc < 1 ) return -EC_NOT_FOUND;
11315:
11316:     if ( ID_IDI48 != board_id ) return -EC_BOARD_TYPE;
11317:
11318:     error_code = -EC_SYNTAX;
11319:
11320:
11321:     strtol( argv[0], &endptr, 0 );
11322:     if ( argv[0] != endptr )
11323:     {
11324:         error_code = (* cmd_idi48[0].cmd_fnc )( board_id, argc, argv );
11325:     }
11326:     else
11327:     {
11328:         index = 0;

```

```
11329:     while ( NULL != cmd_idi48[index].cmd_fnc )
11330:     {
11331:         if ( 0 == strcmpi( cmd_idi48[index].name, argv[0] ) )
11332:         {
11333:             argv_new = &(argv[1]);
11334:             argc_new = argc - 1;
11335:             if ( 0 == argc_new ) argv_new = NULL;
11336:             error_code = (* cmd_idi48[index].cmd_fnc )( board_id, argc_new, argv_new );
11337:             break;
11338:         }
11339:         index++;
11340:     }
11341: }
11342: return error_code;
11343: }
11344:
11345:
11346:
11347:
11348:
11349:
11350:
11351:
11352:
11353: static int CMD__Trace_File( int board_id, int argc, char * argv[] )
11354: {
11355:     int error_code;
11356:     size_t index;
11357:     BOOL not_done;
11358:     struct board_dataset * dataset;
11359:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
11360:
11361:     error_code = SUCCESS;
11362:     if ( argc < 1 )
11363:     {
11364:         printf( "file_name      = " );
11365:         if ( 0 == strlen( dataset->trace.filename ) ) printf( "%s\n", IO_TRACE_DUMP_FILE_NAME );
11366:         else
11367:             printf( "%s\n", dataset->trace.filename );
11368:     }
11369:     else
11370:     {
11371:         index      = 0;
11372:         not_done = true;
11373:         do
11374:         {
11375:             dataset->trace.filename[index] = argv[0][index];
11376:             if ( '\0' == argv[0][index] )
11377:             {
11378:                 not_done = false;
11379:             }
11380:             else if ( index >= ( IO_TRACE_FILE_NAME_SIZE - 2 ) )
11381:             {
11382:                 not_done = false;
11383:                 index++;
11384:                 dataset->trace.filename[index] = '\0';
11385:             }
11386:             index++;
11387:             dataset->trace.filename[index] = '\0';
11388:         }
11389:     }
11390:     else
11391:     {
11392:         index++;
11393:     }
11394: } while ( true == not_done );
11395: }
```

```
11396:     return error_code;
11397: }
11398:
11404: static int CMD__Trace_Stop( int board_id, int argc, char * argv[] )
11405: {
11406:     int error_code;
11407:     struct board_dataset * dataset;
11408:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
11409:
11410:     error_code = SUCCESS;
11411:     if ( argc < 1 )
11412:     {
11413:         printf( "stop      = " );
11414:         printf( "%s\n", global_io_trace_stop_name[dataset->trace.stop] );
11415:
11424:     }
11425: else
11426:     {
11427:         int index = 0;
11428:         error_code = -EC_PARAMETER;
11429:         while ( NULL != global_io_trace_stop_name[index] )
11430:         {
11431:             if ( 0 == strcmpi( global_io_trace_stop_name[index], argv[0] ) )
11432:             {
11433:                 dataset->trace.stop = (enum IO_TRACE_STOP_ENUM) index;
11434:                 error_code = SUCCESS;
11435:                 break;
11436:             }
11437:             index++;
11438:         }
11439:
11457:     }
11458:     return error_code;
11459: }
11460:
11466: static int CMD__Trace_Start( int board_id, int argc, char * argv[] )
11467: {
11468:     int error_code;
11469:     struct board_dataset * dataset;
11470:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
11471:
11472:     error_code = SUCCESS;
11473:     if ( argc < 1 )
11474:     {
11475:         printf( "start      = " );
11476:         printf( "%s\n", global_io_trace_start_name[dataset->trace.start] );
11477:
11486:     }
11487: else
11488:     {
11489:         int index = 0;
11490:         error_code = -EC_PARAMETER;
11491:         while ( NULL != global_io_trace_start_name[index] )
11492:         {
11493:             if ( 0 == strcmpi( global_io_trace_start_name[index], argv[0] ) )
11494:             {
11495:                 dataset->trace.start = (enum IO_TRACE_START_ENUM) index;
11496:                 error_code = SUCCESS;
11497:                 break;
11498:             }
11499:             index++;
11500:         }
11501:
11519:     }
```

```
11520:     return error_code;
11521: }
11522:
11526: static struct command_line_board cmd_trace[] =
11527: {
11528:     { NULL, CMD__Trace_Start, "start", "begin trace on: < " IO_TRACE_START_DEFINITION(
11529:         IO_TRACE_START_STOP_EXTRACT_NAME_HELP_LIST ) ">" },
11530:     { NULL, CMD__Trace_Stop, "stop", "stop trace on: < " IO_TRACE_STOP_DEFINITION(
11531:         IO_TRACE_START_STOP_EXTRACT_NAME_HELP_LIST ) ">" },
11532:     { NULL, CMD__Trace_File, "file", "file_name up to 32-characters" },
11533:     { NULL, NULL, NULL, NULL }
11534: };
11535:
11537: static int CMD__Main_Trace( int board_id, int argc, char* argv[] )
11538: {
11539:     int error_code;
11540:     int index;
11541:     int argc_new;
11542:     char ** argv_new;
11543:     (void) board_id;
11544:
11545:     error_code = -EC_SYNTAX;
11546:
11547:     if ( ( argc < 1 ) || ( NULL == argv ) )
11548:     {
11549:         index = 0;
11550:         while ( NULL != cmd_trace[index].cmd_fnc )
11551:         {
11552:             printf( "trace." );
11553:             argv_new = &(argv[1]);
11554:             argc_new = argc - 1;
11555:             if ( 0 == argc_new ) argv_new = NULL;
11556:             error_code = (* cmd_trace[index].cmd_fnc )( board_id, argc_new, argv_new );
11557:             index++;
11558:         }
11559:     }
11560: else
11561:     {
11562:         index = 0;
11563:         while ( NULL != cmd_trace[index].cmd_fnc )
11564:         {
11565:             if ( 0 == strcmpi( cmd_trace[index].name, argv[0] ) )
11566:             {
11567:                 argv_new = &(argv[1]);
11568:                 argc_new = argc - 1;
11569:                 if ( 0 == argc_new ) argv_new = NULL;
11570:                 error_code = (* cmd_trace[index].cmd_fnc )( board_id, argc_new, argv_new );
11571:                 break;
11572:             }
11573:             index++;
11574:         }
11575:     }
11576:     return error_code;
11577: }
11578:
11596: int Command_Line_Register_Transaction( int board_id, int argc, char* argv[] )
11597: {
11598:     int error_code;
11599:     BOOL valid_bank_reg;
11600:     int index;
11601:     struct reg_definition * definition;
11602:
11603:
11604:     definition = ( struct reg_definition * ) board_definition[board_id].definition;
```

```

11605:
11606:
11607: if ( argc < 1 ) return -EC_NOT_FOUND;
11608:
11609: error_code = Register_Acronym_To_Row( board_id, argv[0], &index );
11610: if ( SUCCESS != error_code ) return error_code;
11611:
11612:
11613: valid_bank_reg = false;
11614: switch( board_id )
11615: {
11616: case ID_IDI48: if ( IDI_BANK == definition[index].symbol ) valid_bank_reg = true;
11617: break;
11618: case ID_IDO48: if ( IDO_BANK == definition[index].symbol ) valid_bank_reg = true;
11619: break;
11620: }
11621:
11622: if ( valid_bank_reg )
11623: {
11624: if ( argc < 2 )
11625: {
11626: uint8_t value;
11627: error_code = IO_Read_U8( board_id, __LINE__, definition[index].symbol, &value );
11628: if ( SUCCESS == error_code )
11629: {
11630: printf( "RD: %s=%s (0x%02X)\n", definition[index].acronym, Bank_Symbol_To_Name(
11631: board_id, ((int) value) ), value );
11632: }
11633: }
11634: else
11635: {
11636: uint8_t value;
11637: int bank;
11638:
11639: error_code = Bank_Name_To_Symbol( board_id, argv[1], &bank );
11640: if ( SUCCESS != error_code )
11641: {
11642: value = (uint8_t) strtol( argv[1], NULL, 0 );
11643: }
11644: else
11645: {
11646: value = (uint8_t) bank;
11647: }
11648: error_code = IO_Write_U8( board_id, __LINE__, definition[index].symbol, value );
11649: if ( SUCCESS == error_code )
11650: {
11651: printf( "WR: %s=0x%02X\n", definition[index].acronym, value );
11652: }
11653: }
11654: }
11655: else
11656: {
11657: if ( argc < 2 )
11658: {
11659: uint8_t value;
11660: error_code = IO_Read_U8( board_id, __LINE__, definition[index].symbol, &value );
11661: if ( SUCCESS == error_code )
11662: {
11663: printf( "RD: %s=0x%02X\n", definition[index].acronym, value );
11664: }
11665: }
11666: else
11667: {
11668: uint8_t value;
11669: value = (uint8_t) strtol( argv[1], NULL, 0 );

```

```
11670:     error_code = IO_Write_U8( board_id, __LINE__, definition[index].symbol, value );
11671:     if ( SUCCESS == error_code )
11672:     {
11673:         printf( "WR: %s=0x%02X\n", definition[index].acronym, value );
11674:     }
11675: }
11676: }
11677:
11678:     return SUCCESS;
11679: }
11680:
11681:
11682: static int CMD__Main_IO_Behavior( int board_id, int argc, char * argv[] )
11683: {
11684:     struct board_dataset * dataset;
11685:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
11686:
11687:     if ( ( argc < 1 ) || ( NULL == argv ) )
11688:     {
11689:         printf( "IO Simulate      = %s\n", dataset->io_simulate ? "true" : "false" );
11690:         printf( "IO Report       = %s\n", dataset->io_report   ? "true" : "false" );
11691:
11692:     else if ( argc > 1 )
11693:     {
11694:         if ( 0 == strcmpi( "simulate", argv[0] ) )
11695:         {
11696:             dataset->io_simulate = String_To_Bool( argv[1] );
11697:         }
11698:         else if ( 0 == strcmpi( "report", argv[0] ) )
11699:         {
11700:             dataset->io_report = String_To_Bool( argv[1] );
11701:         }
11702:     }
11703:     else
11704:     {
11705:         printf( "OK\n" );
11706:
11707:     }
11708:
11709:
11710: }
11711: }
11712: }
11713: else
11714: {
11715:     if ( 0 == strcmpi( "simulate", argv[0] ) )
11716:     {
11717:         printf( "IO Simulate      = %s\n", dataset->io_simulate ? "true" : "false" );
11718:     }
11719:     else if ( 0 == strcmpi( "report", argv[0] ) )
11720:     {
11721:         printf( "IO Report       = %s\n", dataset->io_report   ? "true" : "false" );
11722:     }
11723: }
11724: return SUCCESS;
11725: }
11726:
11727: static int CMD__Main_Base( int board_id, int argc, char * argv[] )
11728: {
11729:     struct board_dataset * dataset;
11730:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
11731:
11732:     if ( ( argc < 1 ) || ( NULL == argv ) )
11733:     {
11734:         printf( "base          = 0x%04X\n", dataset->base_address );
11735:     }
11736:     else
11737:     {
11738:         dataset->base_address = (uint16_t) strtol( argv[0], NULL, 0 );
11739:
```

```

11744:     printf( "OK\n" );
11745: }
11746: return SUCCESS;
11747: }
11748:
11754: static int CMD_Main_Irq_Number( int board_id, int argc, char * argv[] )
11755: {
11756:     struct idi_dataset * idi_dataset = NULL;
11757:
11758:     switch( board_id )
11759:     {
11760:         case ID_IDI48:
11761:             idi_dataset = ( struct idi_dataset * ) board_definition[board_id].dataset;
11762:             break;
11763:     }
11764:
11765:     if ( ( argc < 1 ) || ( NULL == argv ) )
11766:     {
11767:         if ( NULL == idi_dataset ) return SUCCESS;
11768: #if defined( __MSDOS__ )
11769:         printf( "irq                = %u\n", idi_dataset->irq_number );
11770: #else
11771:         printf( "irq                = %u\n", idi_dataset->irq_number );
11772: #endif
11773:     }
11774:     else
11775:     {
11776:         if ( NULL == idi_dataset ) return -EC_BOARD_TYPE;
11777:         idi_dataset->irq_number = ( unsigned int ) strtol( argv[0], NULL, 0 );
11778:         printf( "OK\n" );
11779:     }
11780:     return SUCCESS;
11781: }
11782:
11791: static int CMD_Main_I_Count( int board_id, int argc, char * argv[] )
11792: {
11793:     struct idi_dataset * idi_dataset = NULL;
11794:
11795:     switch( board_id )
11796:     {
11797:         case ID_IDI48:
11798:             idi_dataset = ( struct idi_dataset * ) board_definition[board_id].dataset;
11799:             break;
11800:     }
11801:
11802:     if ( ( argc < 1 ) || ( NULL == argv ) )
11803:     {
11804:         if ( NULL == idi_dataset ) return SUCCESS;
11805: #if defined( __MSDOS__ )
11806:         printf( "iqty                = %d\n", idi_dataset->irq_quantity );
11807: #else
11808:         printf( "iqty                = %lu\n", idi_dataset->irq_quantity );
11809: #endif
11810:     }
11811:     else
11812:     {
11813:         if ( NULL == idi_dataset ) return -EC_BOARD_TYPE;
11814:         idi_dataset->irq_quantity = (size_t) strtol( argv[0], NULL, 0 );
11815:         printf( "OK\n" );
11816:     }
11817:     return SUCCESS;
11818: }
11819:
11820: #if ( 0 )

```

```
11821:  
11822: static int CMD__Main_Init_Reg( int board_id, int argc, char * argv[] )  
11823: {  
11824:     int index;  
11825:     int error_code;  
11826:     struct board_dataset * dataset;  
11827:     struct reg_definition * definition;  
11828:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;  
11829:     definition = ( struct reg_definition * ) board_definition[board_id].definition;  
11830:     if ( argc < 1 )  
11831:     {  
11832:         for ( index = 0; index < REGS_INIT_QTY; index++ )  
11833:         {  
11834:             printf ( "reg_init[%2d]: ", index );  
11835:             if ( dataset->reg_init[index].used )  
11836:             {  
11837:                 if ( IDI_UNDEFINED != dataset->reg_init[index].location )  
11838:                 {  
11839:                     row = REG_LOCATION_LOGICAL_GET( location );  
11840:                     printf( "reg = %s, value = 0x%02X", definition[row].acronym, value );  
11841:                 }  
11842:                 else  
11843:                 {  
11844:                     printf( "address = 0x%04X, value = 0x%02X", address, value );  
11845:                 }  
11846:             }  
11847:             else  
11848:             {  
11849:                 printf( "unused" );  
11850:             }  
11851:         }  
11852:     }  
11853: }  
11854: else  
11855: {  
11856:     printf( "unused" );  
11857: }  
11858: }  
11859: printf( "\n" );  
11860: }  
11861: else if ( argc > 1 )  
11862: {  
11863:     for ( index = 0; index < REGS_INIT_QTY; index++ )  
11864:     {  
11865:         if ( IDI_UNDEFINED != dataset->reg_init[index].location )  
11866:         {  
11867:             if ( SUCCESS == Register_Acronym_To_Row( board_id, argv[0], &index ) )  
11868:             {  
11869:             }  
11870:             }  
11871:         else  
11872:         {  
11873:         }  
11874:     }  
11875: }  
11876: }  
11877: }  
11878:  
11879: if ( argc > 0 )  
11880: {  
11881:  
11882:  
11883: }  
11884:  
11885: return SUCCESS;  
11886: }  
11887: #endif  
11888:  
11894: static int CMD__Main_FRAM_CS( int board_id, int argc, char * argv[] )  
11895: {
```

```

11896: struct board_dataset * dataset;
11897: dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
11898:
11899: if ( ( argc < 1 ) || ( NULL == argv ) )
11900: {
11901:     printf( "framcs          = " );
11902:     if ( true == dataset->fram_cs_default ) printf( "1" );
11903:     else                                printf( "0" );
11904:     printf( "  (FRAM SPI Chip Select)\n" );
11905: }
11906: else
11907: {
11908:     dataset->fram_cs_default = String_To_Bool( argv[0] );
11909:     printf( "OK\n" );
11910: }
11911: return SUCCESS;
11912: }
11913:
11914: static int CMD__Main_AnalogStick_CS( int board_id, int argc, char * argv[] )
11915: {
11916:     struct board_dataset * dataset;
11917:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
11918:
11919:     if ( ( argc < 1 ) || ( NULL == argv ) )
11920:     {
11921:         printf( "aics            = " );
11922:         if ( true == dataset->as_cs_default ) printf( "1" );
11923:         else                                printf( "0" );
11924:         printf( "  (Analog Stick SPI Chip Select)\n" );
11925:     }
11926:     else
11927:     {
11928:         dataset->as_cs_default = String_To_Bool( argv[0] );
11929:         printf( "OK\n" );
11930:     }
11931:     return SUCCESS;
11932: }
11933:
11934: static int CMD__Main_Mode_Jumpers( int board_id, int argc, char * argv[] )
11935: {
11936:     struct board_dataset * ds = NULL;
11937:
11938:     switch( board_id )
11939:     {
11940:         case ID_IDI48:
11941:             ds = ( struct board_dataset * ) board_definition[board_id].dataset;
11942:             break;
11943:         case ID_IDO48:
11944:             ds = ( struct board_dataset * ) board_definition[board_id].dataset;
11945:             break;
11946:     }
11947:
11948:     if ( ( argc < 1 ) || ( NULL == argv ) )
11949:     {
11950:         if ( NULL == ds ) return SUCCESS;
11951:         printf( "mode           = " );
11952:         switch( ds->mode_jumpers )
11953:         {
11954:             case MODE_JUMPERS_0: printf( "0 (legacy)" ); break;
11955:             case MODE_JUMPERS_1: printf( "1           " ); break;
11956:             case MODE_JUMPERS_2: printf( "2           " ); break;
11957:             case MODE_JUMPERS_3: printf( "3           " ); break;
11958:         }
11959:     }
11960:
11961:     printf( ". Assumed mode jumper settings.\n" );

```

```
11973: }
11974: else
11975: {
11976:     if ( NULL == ds ) return -EC_BOARD_TYPE;
11977:     if ( 0 == strcasecmp( "legacy", argv[0] ) )
11978:     {
11979:         ds->mode_jumpers = MODE_JUMPERS_0;
11980:     }
11981: else
11982: {
11983:     ds->mode_jumpers = (MODE_JUMPERS_ENUM) strtol( argv[0], NULL, 0 );
11984: }
11985: printf( "OK\n" );
11986: }
11987: return SUCCESS;
11988: }
11989:
11993: static struct command_line_board cmd_set[] =
11994: {
11995: { NULL,    CMD_Main_Base,      "base",      "params: [<address>]" },
11996: { NULL,    CMD_Main_IO_Behavior, "io",        "params: [<simulate>/<report>]" },
11997: { NULL,    CMD_Main_Irq_Number, "irq",       "params: [<irq_number 0 to 15>]" },
11998: { NULL,    CMD_Main_I_Count,   "igty",      "params: [<number>]. Number of interrupts." },
11999: { NULL,    CMD_Main_FRAM_CS,   "framcs",    "params: [<0/1>]. FRAM SPI chip select channel." },
12000: { NULL,    CMD_Main_AnalogStick_CS, "aics",     "params: [<0/1>]. Analog Stick SPI chip select channel." },
12001: { NULL,    CMD_AS_Parameter,   "as",        "Analog Stick register or parameter read/write" },
12002: { cmd_trace, CMD_Main_Trace,   "trace",     "params: [<start/stop/file>]." },
12003: { NULL,    CMD_Main_Mode_Jumpers, "mode",     "params: [<0/1/2/3>]. Mode jumper settings" },
12004: { NULL,    NULL,           NULL,      NULL },
12005: };
12006:
12012: int Command_Line_Set( int board_id, int argc, char* argv[] )
12013: {
12014:     int error_code;
12015:     int index;
12016:     int argc_new;
12017:     char ** argv_new;
12018:     struct board_dataset * dataset;
12019:     (void) board_id;
12020:
12021:     dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
12022:     error_code = -EC_SYNTAX;
12023:
12024:     if ( argc < 1 )
12025:     {
12026:         index = 0;
12027:         while ( NULL != cmd_set[index].cmd_fnc )
12028:         {
12029:             argv_new = &(argv[1]);
12030:             argc_new = argc - 1;
12031:             if ( 0 == argc_new ) argv_new = NULL;
12032:             dataset->set__suppress_io_activity = true;
12033:             error_code = (* cmd_set[index].cmd_fnc )( board_id, argc_new, argv_new );
12034:             dataset->set__suppress_io_activity = false;
12035:             index++;
12036:         }
12037:     }
12038: else
```

```
12039: {
12040:     index = 0;
12041:     while ( NULL != cmd_set[index].cmd_fnc )
12042:     {
12043:         if ( 0 == strcmpi( cmd_set[index].name, argv[0] ) )
12044:         {
12045:             argv_new = &(argv[1]);
12046:             argc_new = argc - 1;
12047:             if ( 0 == argc_new ) argv_new = NULL;
12048:             dataset->set__suppress_io_activity = true;
12049:             error_code = (* cmd_set[index].cmd_fnc )( board_id, argc_new, argv_new );
12050:             dataset->set__suppress_io_activity = false;
12051:             break;
12052:         }
12053:         index++;
12054:     }
12055: }
12056: return error_code;
12057: }
12058:
12064: int Command_Line_Dump( int board_id, int argc, char* argv[] )
12065: {
12066:     int error_code;
12067:     FILE * fd_out;
12068:
12069:
12070:
12071:     if ( argc > 0 )
12072:     {
12073:         fd_out = fopen( argv[0], "w" );
12074:         if ( NULL == fd_out ) fd_out = stdout;
12075:     }
12076:     else
12077:     {
12078:         fd_out = stdout;
12079:     }
12080:     error_code = Register_Report_CSV( board_id, fd_out );
12081:
12082:     if ( (argc > 0) && (NULL != fd_out) && (stdout != fd_out) )
12083:     {
12084:         fclose( fd_out );
12085:     }
12086:     return error_code;
12087: }
12088:
12089:
12099: int Command_Line_IO_Write( int board_id, int argc, char* argv[] )
12100: {
12101:     size_t index;
12102:     BOOL address_captured;
12103:     int error_code;
12104:     int address;
12105:     uint8_t value;
12106:     BOOL auto_index;
12107:     (void) board_id;
12108:
12109:     if ( argc < 2 ) return -EC_NOT_FOUND;
12110:
12111:     address_captured = false;
12112:     auto_index = true;
12113:     for ( index = 0; index < argc; index++ )
12114:     {
12115:         if ( ( 'x' == argv[index][0] ) || ( 'X' == argv[index][0] ) )
12116:         {
```

```
12117:     auto_index = !auto_index;
12118: }
12119: else
12120: {
12121:     if ( false == address_captured )
12122:     {
12123:         address = (int) strtol( argv[index], NULL, 0 );
12124:         address_captured = true;
12125:
12126:     }
12127: else
12128: {
12129:     value = (uint8_t) strtol( argv[index], NULL, 0 );
12130:     error_code = IO_Write_U8_Port( NULL, __LINE__, address, value );
12131:     if ( SUCCESS != error_code ) return error_code;
12132:     if ( true == auto_index ) address++;
12133: }
12134: }
12135: }
12136: printf( "OK\n" );
12137: return error_code;
12138: }
12139:
12140: int Command_Line_IO_Read( int board_id, int argc, char* argv[] )
12141: {
12142:     size_t index;
12143:     BOOL count_capture;
12144:     BOOL address_captured;
12145:     BOOL auto_index;
12146:     int error_code;
12147:     int address;
12148:     int count;
12149:     uint8_t value;
12150:     (void) board_id;
12151:
12152:     if ( argc < 1 ) return -EC_NOT_FOUND;
12153:
12154:     count = 1;
12155:     error_code = SUCCESS;
12156:     count_capture = false;
12157:     address_captured = false;
12158:     auto_index = true;
12159:     for ( index = 0; index < argc; index++ )
12160:     {
12161:         if ( ( 'c' == argv[index][0] ) || ( 'C' == argv[index][0] ) )
12162:         {
12163:             count_capture = true;
12164:         }
12165:         else if ( ( 'x' == argv[index][0] ) || ( 'X' == argv[index][0] ) )
12166:         {
12167:             auto_index = !auto_index;
12168:         }
12169:     }
12170:     if ( true == count_capture )
12171:     {
12172:         count = (int) strtol( argv[index], NULL, 0 );
12173:         count_capture = false;
12174:     }
12175:     else
12176:     {
12177:         if ( false == address_captured )
12178:         {
12179:             if ( true == count_capture )
12180:             {
12181:                 count = (int) strtol( argv[index], NULL, 0 );
12182:                 count_capture = false;
12183:             }
12184:         }
12185:     }
12186: }
```

```
12190:     address = (int) strtol( argv[index], NULL, 0 );
12191:     address_captured = true;
12192:     printf( "0x%04X:", address );
12193: }
12194: else
12195: {
12196:
12197:
12198: }
12199: }
12200: }
12201: }
12202:
12203: for ( index = 0; index < count; index++ )
12204: {
12205:     error_code = IO_Read_U8_Port( NULL, __LINE__, address, &value );
12206:     if ( SUCCESS != error_code ) return error_code;
12207:     printf( " 0x%02X", value );
12208:     if ( true == auto_index ) address++;
12209: }
12210:
12211: printf( "\n" );
12212: return error_code;
12213: }
12214:
12215: int Command_Line_Wait( int board_id, int argc, char* argv[] )
12216: {
12217:     (void) board_id;
12218:     (void) argc;
12219:     (void) argv;
12220:
12221:
12222:
12223:
12224:     if ( global_loop_command ) return SUCCESS;
12225:     global_loop_command = true;
12226:     return SUCCESS;
12227: }
12228:
12229:
12230: static void Help_Pause_Helper( BOOL skip_pause, size_t * line )
12231: {
12232:     if ( true == skip_pause ) return;
12233:     if ( *line > 20 )
12234:     {
12235:         printf( "Press any key for more....." );
12236:         while ( !Character_Get( NULL ) ) { }
12237:         printf( "\r" );
12238:         *line = 0;
12239:     }
12240: }
12241:
12242: static void Help_Output( BOOL skip_pause,
12243:                         int depth,
12244:                         size_t * line,
12245:                         struct command_line_board * table,
12246:                         FILE * out
12247:                         )
12248:
12249: {
12250:     size_t row;
12251:     int index;
12252:     const char * prefix = { "        " };
12253:
12254:     row = 0;
12255:     while ( NULL != table[row].name )
12256:     {
12257:         if ( 0 == depth )
```

```
12264: {
12265:     fprintf( out, "%-4s", table[row].name );
12266: }
12267: else
12268: {
12269:     for ( index = 0; index < depth; index++ ) fprintf( out, "%s", prefix );
12270:     fprintf( out, "%-8s", table[row].name );
12271: }
12272: fprintf( out, " - %s\n", table[row].help );
12273: *line += 1;
12274: #if defined( __MSDOS__ )
12275:     Help_Pause_Helper( skip_pause, line );
12276: #endif
12277: if ( NULL != table[row].link )
12278: {
12279:     Help_Output( skip_pause, depth + 1, line, table[row].link, out );
12280:     fprintf( out, "\n" );
12281:     *line += 1;
12282: }
12283: row++;
12284: }
12285: }
12286:
12291: static void IDI_Help( size_t * line, FILE * out )
12292: {
12293:     fprintf( out, "IDI48 board supported\n" );
12294:     fprintf( out, " Examples:\n" );
12295:     fprintf( out, "    idi dig0          <-- reports the digital input group 0 port.\n" );
12296:     fprintf( out, "    idi loop dig0      <-- loops until key pressed\n" );
12297:     fprintf( out, "    idi irq loop dig0 <-- loops while counting interrupts or until iqty
reached\n" );
12298:     fprintf( out, "    idi irq dig0       <-- does command once and uses interrupts for
short time\n" );
12299:     *line += 7;
12300: }
12301:
12306: static void IDO_Help( size_t * line, FILE * out )
12307: {
12308:     fprintf( out, "IDO48 board supported\n" );
12309:     fprintf( out, " Examples:\n" );
12310:     fprintf( out, "    ido dog0          <-- reports the digital output group 0 port.\n" );
12311:     fprintf( out, "    ido loop dog0      <-- loops until key pressed\n" );
12312:     *line += 5;
12313: }
12314:
12315:
12316: static struct command_line_board cmd_top[];
12317: static struct command_line cmd_prefix[];
12318:
12319:
12320:
12325: static void Help( BOOL skip_pause, FILE * out )
12326: {
12327:     size_t line;
12328:     int top;
12329:
12330:     line = 0;
12331:     fprintf( out, "\n" );
12332:     fprintf( out, "STDBus General Test Code\n" );
12333:     fprintf( out, "Apex Embedded Systems\n" );
12334:     fprintf( out, "SVN software revision: " SVN_REV "\n" );
12335:     fprintf( out, "Supporting:\n" );
12336:     line += 5;
12337:
```

```
12338: IDI_Help( &line, out );
12339: IDO_Help( &line, out );
12340:
12341: fprintf( out, "\n" );
12342: fprintf( out, "help - outputs help information\n" );
12343: line += 2;
12344:
12345: top = 0;
12346: while ( NULL != cmd_prefix[top].name )
12347: {
12348:     fprintf( out, "\n" );
12349:     fprintf( out, "%-4s - %s\n", cmd_prefix[top].name, cmd_prefix[top].help );
12350:     top++;
12351:     line += 2;
12352:     Help_Pause_Helper( skip_pause, &line );
12353: }
12354:
12355: Help_Output( skip_pause, 0, &line, (struct command_line_board *) cmd_top , out );
12356: fprintf( out, "\n" );
12357: }
12358:
12362: int Command_Line_Help( int board_id, int argc, char* argv[] )
12363: {
12364:     FILE * out;
12365:     FILE * fd = NULL;
12366:     BOOL skip_pause = false;
12367:     (void) board_id;
12368:
12369:     out = stdout;
12370:     if ( argc > 0 )
12371:     {
12372:         fd = fopen( argv[0], "w" );
12373:         if ( NULL != fd )
12374:         {
12375:             out = fd; skip_pause = true;
12376:         }
12377:     }
12378:     Help( skip_pause, out );
12379:     if ( NULL != fd ) fclose( fd );
12380:
12381:     return SUCCESS;
12382: }
12383:
12387: int FPGA_Date_Time_Information( int board_id, uint32_t * date_svn, uint32_t * date_compilation )
12388: {
12389:     int location_data;
12390:     int location_index;
12391:     int state;
12392:     size_t index;
12393:     size_t count;
12394:     BOOL fpga_data_available;
12395:     const uint8_t index_test_list[] = { 0x05, 0x0A, 0x00 };
12396:     uint8_t scratch[16] = { 0x00 };
12397:     uint8_t block_offset;
12398:     uint8_t block_type;
12399:     uint32_t scratch_u32;
12400: #define FPGA_DATE_SIZE 4
12401:     struct board_dataset * dataset;
12402:     dataset = (struct board_dataset *) board_definition[board_id].dataset;
12403:
12404:     if ( MODE_JUMPERS_0 == dataset->mode_jumpers )
12405:     {
12406:         return -EC_MODE_LEGACY;
```

```
12407: }
12408:
12409:
12410: location_data      = board_definition[board_id].fpga_data_symbol;
12411: location_index     = board_definition[board_id].fpga_index_symbol;
12412: fpga_data_available = true;
12413:
12414: count      = 0;
12415: while ( 0x00 != index_test_list[count] )
12416: {
12417:   IO_Write_U8( board_id, __LINE__, location_index, index_test_list[count] );
12418:   IO_Read_U8( board_id, __LINE__, location_index, &(scratch[count]) );
12419:   count++;
12420: }
12421: for ( index = 0; index < count; index++ )
12422: {
12423:   if ( index_test_list[index] != scratch[index] )
12424:   {
12425:     fpga_data_available = false;
12426:     break;
12427:   }
12428: }
12429:
12430: if ( false == fpga_data_available )
12431: {
12432:   *date_svn = 0;
12433:   *date_compilation = 0;
12434:   return SUCCESS;
12435: }
12436:
12437: state      = 0;
12438: index      = 0;
12439: block_offset = 0;
12440: while( 3 != state )
12441: {
12442:   switch ( state )
12443:   {
12444:     case 0:
12445:       IO_Write_U8( board_id, __LINE__, location_index, (uint8_t) index );
12446:       index++;
12447:       IO_Read_U8( board_id, __LINE__, location_data, &block_offset );
12448:
12449:       IO_Write_U8( board_id, __LINE__, location_index, (uint8_t) index );
12450:       index++;
12451:       IO_Read_U8( board_id, __LINE__, location_data, &block_type );
12452:
12453:       switch( block_type )
12454:       {
12455:         case 's':
12456:         case 'S':
12457:           state = 1;  count = 0;
12458:           break;
12459:         case 'c':
12460:         case 'C':
12461:           state = 2;  count = 0;
12462:           break;
12463:         case 0x00:
12464:         case 0xFF:
12465:           state = 3;
12466:           count = 0;
12467:           break;
12468:         default:
12469:           break;
12470:       }
```

```

12471:     break;
12472: case 1:
12473:     IO_Write_U8( board_id, __LINE__, location_index, (uint8_t) index );
12474:     index++; count++;
12475:     IO_Read_U8( board_id, __LINE__, location_data, &(scratch[FPGA_DATE_SIZE - count]) );
12476: }
12477: if ( FPGA_DATE_SIZE == count )
12478: {
12479:     size_t i;
12480:     scratch_u32 = 0;
12481:     for ( i = 0; i < FPGA_DATE_SIZE; i++ )
12482:     {
12483:         scratch_u32 = ( scratch_u32 << 8 ) + ( (uint32_t) scratch[i] );
12484:     }
12485:     *date_svn = scratch_u32;
12486:     state = 0;
12487: }
12488: break;
12489: case 2:
12490:     IO_Write_U8( board_id, __LINE__, location_index, (uint8_t) index );
12491:     index++; count++;
12492:     IO_Read_U8( board_id, __LINE__, location_data, &(scratch[FPGA_DATE_SIZE - count]) );
12493: }
12494: if ( FPGA_DATE_SIZE == count )
12495: {
12496:     size_t i;
12497:     scratch_u32 = 0;
12498:     for ( i = 0; i < FPGA_DATE_SIZE; i++ )
12499:     {
12500:         scratch_u32 = ( scratch_u32 << 8 ) + ( (uint32_t) scratch[i] );
12501:     }
12502:     *date_compilation = scratch_u32;
12503:     state = 0;
12504: }
12505: }
12506: return SUCCESS;
12507: #undef FPGA_DATE_SIZE
12508: }
12509:
12510:
12528:
12529:
12530:
12531:
12535: int Command_Line_FPGA( int board_id, int argc, char* argv[] )
12536: {
12537:     int error_code;
12538:     FILE * out;
12539:     uint32_t date_svn;
12540:     uint32_t date_compilation;
12541:     (void) argc;
12542:     (void) argv;
12543:
12544:     error_code = FPGA_Date_Time_Information( board_id, &date_svn, &date_compilation );
12545:     if ( error_code < SUCCESS ) goto Command_Line_FPGA_Exit;
12546:
12547:     out = stdout;
12548:
12549:     fprintf( out, "FPGA: " );
12550:
12551:     if ( 0 == date_svn )
12552:     {

```

```
12553:     fprintf( out, "SVN = unavailable" );
12554: }
12555: else
12556: {
12557: #if defined( __MSDOS__ )
12558:     fprintf( out, "SVN =      %08lu", date_svn );
12559: #else
12560:     fprintf( out, "SVN =      %08u", date_svn );
12561: #endif
12562: }
12563:
12564: if ( 0 == date_compilation )
12565: {
12566:     fprintf( out, ", Compilation = unavailable\n" );
12567: }
12568: else
12569: {
12570: #if defined( __MSDOS__ )
12571:     fprintf( out, ", Compilation = %08lu\n", date_compilation );
12572: #else
12573:     fprintf( out, ", Compilation = %08u\n", date_compilation );
12574: #endif
12575: }
12576:
12577: Command_Line_FPGA_Exit:
12578:     return error_code;
12579: }
12580:
12581: #define CMD_MAIN_LIST(_)
12582:     \
12583:     _( NULL,    Command_Line_Dump,      "dump",    "Register information in CSV format",   true )
12584: \
12585:     _(
12586:         cmd_set,    Command_Line_Set,      "set",      "Set/Get main parameters",       true )
12587:     _(
12588:         cmd_spi,    Command_Line_SPI,      "spi",      "SPI related functions",       true )
12589:     _(
12590:         cmd_as,    Command_Line_Analog_Stick, "as",      "Analog Stick related functions",
12591:         true )
12592:     _(
12593:         cmd_fram,    Command_Line_FRAM,      "fram",      "FRAM related functions",       true )
12594:     _(
12595:         cmd_idi48,    Command_Line_IDI48,      "di",      "Digital input related functions",
12596:         false )
12597:     _(
12598:         cmd_ido48,    Command_Line_IDO48,      "do",      "Digital output related functions",
12599:         false )
12600:     _(
12601:         NULL,    Command_Line_IO_Write,      "o",      "unrestricted I/O write <address> <data>",
12602:         false )
12603:     _(
12604:         NULL,    Command_Line_IO_Read,      "i",      "unrestricted I/O read  <address>",
12605:         false )
12606:     _(
12607:         NULL,    Command_Line_Wait,      "wait",      "wait for any key press",       false )
12608:     _(
12609:         NULL,    Command_Line_Help,      "help",      "very brief command summary",   false )
12610:     _(
12611:         NULL,    Command_Line_FPGA,      "fpga",      "fpga compilation and svn dates", false )
12612:     _(
12613:         NULL,    NULL,      NULL,      false )
12614:     _(
12615:         CMD_MAIN_LIST( CMD_MAIN_EXTRACT_COMMANDS )
12616:     );
12617:
```

```

12629: int Command_Line_Main( int board_id, int argc, char* argv[] )
12630: {
12631:     int error_code;
12632:     int index;
12633:     BOOL not_found;
12634:     int argc_new;
12635:     char ** argv_new;
12636:
12637:
12638:     error_code = -EC_SYNTAX;
12639:
12640:     if ( argc < 1 ) return -EC_NOT_FOUND;
12641:
12642:
12643:     not_found = true;
12644:     index = 0;
12645:     while ( NULL != cmd_top[index].cmd_fnc )
12646:     {
12647:         if ( 0 == strcasecmp( cmd_top[index].name, argv[0] ) )
12648:         {
12649:             not_found = false;
12650:             argv_new = &(argv[1]);
12651:             argc_new = argc - 1;
12652:             if ( 0 == argc_new ) argv_new = NULL;
12653:             error_code = (* cmd_top[index].cmd_fnc )( board_id, argc_new, argv_new );
12654:             if ( SUCCESS != error_code ) goto COMMAND_LINE_MAIN_TERMINATE;
12655:             break;
12656:         }
12657:         index++;
12658:     }
12659:
12660:     if ( not_found )
12661:     {
12662:         argv_new = &(argv[0]);
12663:         argc_new = argc - 0;
12664:         error_code = Command_Line_Register_Transaction( board_id, argc_new, argv_new );
12665:         if ( SUCCESS != error_code ) goto COMMAND_LINE_MAIN_TERMINATE;
12666:     }
12667: COMMAND_LINE_MAIN_TERMINATE:
12668:     return error_code;
12669: }
12670:
12675: enum CLM_BTS
12676: {
12677:     CLM_BTS_NORMAL      = 0,
12678:     CLM_BTS_NOT_REQUIRED = 1,
12679:     CLM_BTS IMPLIED      = 2
12680: };
12681:
12682:
12692: static enum CLM_BTS Command_Line_Main__Board_Type_Status( const char * cmd_string, int
* board_id )
12693: {
12694:     int index;
12695:     enum CLM_BTS status;
12696:     index = 0;
12697:     status = CLM_BTS_NORMAL;
12698:     *board_id = ID_BOARD_UNKNOWN;
12699:     while ( NULL != cmd_top[index].cmd_fnc )
12700:     {
12707:
12708:
12709:         if ( 0 == strcasecmp( "do", cmd_string ) )
12710:     {

```

```
12711:     *board_id = ID_IDO48; status = CLM_BTS_IMPLIED;
12712: }
12713: else if ( 0 == strcasecmp( "di", cmd_string ) )
12714: {
12715:     *board_id = ID_IDI48; status = CLM_BTS_IMPLIED;
12716: }
12717:
12718:
12719:
12720: if ( 0 == strcasecmp( cmd_top[index].name, cmd_string ) )
12721: {
12722:     if ( false == cmd_top_board_type_required[index] )
12723:     {
12724:         if ( CLM_BTS_NORMAL == status ) status = CLM_BTS_NOT_REQUIRED;
12725:     }
12726:     return status;
12727: }
12728: index++;
12729: }
12730: return status;
12731: }
12732:
12733: static int Command_Line_Loop_Count( int argc,
12734:                                     char* argv[]
12735:                                     )
12736: {
12737:     int error_code;
12738:
12739:     if ( argc > 0 )
12740:     {
12741:         if ( argv[0][0] >= '0' && argv[0][0] <= '9' )
12742:         {
12743:             global_loop_count = (int) strtol( argv[0], NULL, 0 );
12744:             global_loop_count_counter = global_loop_count;
12745:             error_code = 1;
12746:         }
12747:     }
12748:     else
12749:     {
12750:         error_code = -EC_SYNTAX;
12751:     }
12752: }
12753: }
12754: }
12755: else
12756: {
12757:     error_code = -EC_SYNTAX;
12758: }
12759: return error_code;
12760: }
12761:
12762: static int Command_Line_Loop_Delay( int argc,
12763:                                     char* argv[]
12764:                                     )
12765: {
12766:     int error_code;
12767:
12768:     if ( argc > 0 )
12769:     {
12770:         if ( ( argv[0][0] >= '0' ) && ( argv[0][0] <= '9' ) )
12771:         {
12772:             global_loop_delay_ms = (int) strtol( argv[0], NULL, 0 );
12773:             error_code = 1;
12774:         }
12775:     }
12776:     else
12777:     {
12778:         error_code = -EC_SYNTAX;
12779:     }
12780: }
```

```

12781:    }
12782:  }
12783: else
12784:  {
12785:    error_code = -EC_SYNTAX;
12786:  }
12787: return error_code;
12788: }
12789:
12793: static int Command_Line_Loop_Space( int argc,
12794:                                       char* argv[]
12795:                                       )
12796: {
12797:  (void) argc;
12798:  (void) argv;
12799:
12800:  global_loop_space = true;
12801: return SUCCESS;
12802: }
12803:
12807: static struct command_line cmd_loop[] =
12808: {
12809:  { NULL, Command_Line_Loop_Count,      "count", "loop for N counts" },
12810:  { NULL, Command_Line_Loop_Delay,     "delay", "loop with delay <milliseconds>" },
12811:  { NULL, Command_Line_Loop_Space,     "space", "loop each time space bar pressed" },
12812:  { NULL, NULL,           NULL, NULL }
12813: },
12814: };
12815:
12819: static int Command_Line_Prefix_Loop( int argc,
12820:                                       char* argv[]
12821:                                       )
12822: {
12823:  int error_code;
12824:  int index;
12825:  int ai;
12826:
12827:  int argc_new;
12828:  char ** argv_new;
12829:
12830:  global_loop_command = true;
12831:
12832:  index = 0;
12833:  ai = 0;
12834: while ( NULL != cmd_loop[index].cmd_fnc )
12835: {
12836:  if ( 0 == strcasecmp( cmd_loop[index].name, argv[ai] ) )
12837:  {
12838:
12839:   argv_new = &(argv[ai + 1]);
12840:   argc_new = argc - (ai + 1);
12841:   if ( 0 == argc_new ) argv_new = NULL;
12842:   error_code = (* cmd_loop[index].cmd_fnc )( argc_new, argv_new );
12843:   if ( error_code < 0 ) goto COMMAND_LINE_PREFIX_LOOP_TERMINATE;
12844:   else ai += 1 + error_code;
12845:   index = 0;
12846: }
12847: else
12848: {
12849:   index++;
12850: }
12851: }
12852: return ai;
12853: COMMAND_LINE_PREFIX_LOOP_TERMINATE:

```

```
12854:     return error_code;
12855: }
12856:
12860: static int Command_Line_Prefix_Irq( int argc,
12861:                                     char* argv[ ]
12862:                                     )
12863: {
12864:     (void) argc;
12865:     (void) argv;
12866:
12867:     global_irq.Please_Install_Handler_Request = true;
12868: #if(0)
12869:     switch( board_id )
12870:     {
12871:         case ID_IDI48:
12872:             ( ( struct idi_dataset * )  
board_definition[board_id].dataset->irq.Please_Install_Handler_Request = true;
12873:             break;
12874:         default:
12875:             return -EC_BOARD_TYPE;
12876:     }
12877: #endif
12878:
12879:     return 0;
12880: }
12881:
12885: static int Command_Line_Prefix_Trace( int argc,
12886:                                         char* argv[ ]
12887:                                         )
12888: {
12889:     (void) argc;
12890:     (void) argv;
12891:     global_io_trace_enable = true;
12892:     return SUCCESS;
12893: }
12894:
12895: static struct command_line cmd_prefix[ ] =
12896: {
12897:     { NULL, Command_Line_Prefix_Irq, "irq", "allows commands to handle interrupts" },
12898:     { cmd_loop, Command_Line_Prefix_Loop, "loop", "any command loops until key
12899:     pressed" },
12900:     { NULL, Command_Line_Prefix_Trace, "trace", "enable trace functionality" },
12901:     { NULL, NULL, NULL, NULL }
12902: };
12903:
12904: };
12905:
12906: int Command_Line_Prefix( int argc, char* argv[ ] )
12907: {
12908:     int error_code;
12909:     int index;
12910:     int ai;
12911:     int argc_new;
12912:     char ** argv_new;
12913:
12914:     if ( argc < 1 ) return 0;
12915:
12916:     index = 0;
12917:     ai = 0;
12918:
12919:     if ( argc < 1 ) return 0;
12920:
12921:     index = 0;
12922:     ai = 0;
12923:     while ( NULL != cmd_prefix[index].cmd_fnc )
12924:     {
12925:         if ( 0 == strcasecmp( cmd_prefix[index].name, argv[0] ) )
12926:         {
12927:             argv_new = &(argv[1]);
12928:             argc_new = argc - 1;
```

```
12929:     if ( 0 == argc_new ) argv_new = NULL;
12930:     error_code = (* cmd_prefix[index].cmd_fnc )( argc_new, argv_new );
12931:     if ( error_code < 0 ) goto COMMAND_LINE_PREFIX_TERMINATE;
12932:     else ai = 1 + error_code;
12933:     break;
12934:   }
12935:   index++;
12936: }
12937: return ai;
12938: COMMAND_LINE_PREFIX_TERMINATE:
12939: return error_code;
12940: }
12941:
12942:
12943:
12944:
12945:
12946:
12947:
12948:
12949:
12950:
12951:
12952:
12953:
12954:
12955:
12956:
12957:
12962: static int IDI_Termination( struct idi_dataset * dataset )
12963: {
12964:   FILE * fd;
12965:
12966: #if defined( MAIN_DEBUG )
12967: printf( "Main: IDI_Termination\n" );
12968: #endif
12969:
12970: #if defined( __MSDOS__ )
12971:   IOKern_Resource_Termination();
12972: #endif
12973:   if ( dataset->irq_handler_active )
12974:   {
12975: #if defined( __MSDOS__ )
12976:   printf( " --> irq %u: count = %d\n", dataset->irq_number, dataset->irq_count );
12977: #else
12978:   printf( " --> irq %u: count = %lu\n", dataset->irq_number, dataset->irq_count );
12979: #endif
12980:   }
12981:   dataset->irq_handler_active = false;
12982:   dataset->irq_please_install_handler_request = false;
12983:
12984:
12985:   fd = fopen( IDI_BOARD_INIT_FILE_NAME, "wb" );
12986:   if ( NULL == fd )
12987:   {
12988:
12989: #if defined( MAIN_DEBUG )
12990:   printf("IDI_Termination: fopen() error: %d (%s)\n", errno, strerror(errno) );
12991: #endif
12992:   return -EC_INIT_FILE;
12993: }
12994: else
12995: {
12996: #if defined( MAIN_DEBUG )
12997: printf( "IDI_Termination: sizeof( struct idi_dataset ) = " );
```

```
12998: # if defined( __MSDOS__ )
12999:     printf( "%u\n", sizeof( struct idi_dataset ) );
13000: # else
13001:     printf( "%lu\n", sizeof( struct idi_dataset ) );
13002: # endif
13003: #endif
13004:
13005:         fwrite( dataset, sizeof( struct idi_dataset ), 1, fd );
13006:     fclose( fd );
13007: }
13008: return SUCCESS;
13009: }
13010:
13015: static int IDO_Termination( struct ido_dataset * dataset )
13016: {
13017:     FILE * fd;
13018:
13019: #if defined( MAIN_DEBUG )
13020: printf( "Main: IDO_Termination\n" );
13021: #endif
13022:
13023:
13024:     fd = fopen( IDO_BOARD_INIT_FILE_NAME, "wb" );
13025:     if ( NULL == fd )
13026:     {
13027: #if defined( MAIN_DEBUG )
13028:     printf( "IDO_Termination: fopen() error: %d (%s)\n", errno, strerror(errno) );
13029: #endif
13030:     return -EC_INIT_FILE;
13031: }
13032: else
13033: {
13034: #if defined( MAIN_DEBUG )
13035: printf( "IDO_Termination: sizeof( struct ido_dataset ) = " );
13036: # if defined( __MSDOS__ )
13037:     printf( "%u\n", sizeof( struct ido_dataset ) );
13038: # else
13039:     printf( "%lu\n", sizeof( struct ido_dataset ) );
13040: # endif
13041: #endif
13042:     fwrite( dataset, sizeof( struct ido_dataset ), 1, fd );
13043:     fclose( fd );
13044: }
13045: return SUCCESS;
13046: }
13047:
13055: int Termination( int board_id )
13056: {
13057:     int error_code;
13058:
13059:
13060:
13061:
13062:
13063: #if defined( MAIN_DEBUG )
13064: printf( "Main: Termination\n" );
13065: #endif
13066:
13067:     error_code = IO_Trace_Dump( board_id );
13068:     switch( board_id )
13069:     {
13070:         case ID_IDI48:
13071:             error_code = IDI_Termination( (struct idi_dataset *)
board_definition[board_id].dataset );
```

```

13072:     break;
13073: case ID_IDO48:
13074:     error_code = IDO_Termination( (struct ido_dataset *) 
board_definition[board_id].dataset );
13075:     break;
13076: }
13077: return error_code;
13078: }
13079:
13083: void IDI_Dataset_Defaults( struct idi_dataset * ds )
13084: {
13085:     int index;
13086:
13087:     printf( "Warning - using main IDI48 dataset defaults\n" );
13088:
13089:     memset( ds, 0, sizeof(struct idi_dataset) );
13090:
13091:     ds->base_address      = 0x0110;
13092:
13093:
13094:     for ( index = 0; index < REGS_INIT_QTY; index++ )
13095:     {
13096:         ds->reg_init[index].used      = false;
13097:         ds->reg_init[index].address   = 0;
13098:         ds->reg_init[index].location  = IDI_UNDEFINED;
13099:         ds->reg_init[index].value     = 0x00;
13100:     }
13101: }
13102:
13106: void IDO_Dataset_Defaults( struct ido_dataset * ds )
13107: {
13108:     int index;
13109:
13110:     printf( "Warning - using main IDO48 dataset defaults\n" );
13111:
13112:     memset( ds, 0, sizeof(struct ido_dataset) );
13113:
13114:     ds->base_address      = 0x0130;
13115:
13116:
13117:     for ( index = 0; index < REGS_INIT_QTY; index++ )
13118:     {
13119:         ds->reg_init[index].used      = false;
13120:         ds->reg_init[index].address   = 0;
13121:         ds->reg_init[index].location  = IDO_UNDEFINED;
13122:         ds->reg_init[index].value     = 0x00;
13123:     }
13124: }
13125:
13129: void Signal_Handler( int signal )
13130: {
13131:     struct board_dataset * dataset = ( struct board_dataset * )
board_definition[global_board_id].dataset;
13132:     (void) signal;
13133:     dataset->quit_application = true;
13134:     if ( true == global_io_trace_enable )
13135:     {
13136:         if ( IO_TRACE_STOP_CTRLC == dataset->trace.stop ) dataset->trace.active = false;
13137:     }
13138: }
13139:
13140:
13145: static void IDI_Initialization_Data_Structure_Load( struct idi_dataset * dataset )
13146: {

```

```
13147: FILE * fd;
13148: size_t file_size;
13149:
13150: fd = fopen( IDI_BOARD_INIT_FILE_NAME, "rb" );
13151: if ( NULL == fd )
13152: {
13153:     IDI_Dataset_Defaults( dataset );
13154: #if defined( MAIN_DEBUG )
13155: printf( "IDI_Initialization_Data_Structure_Load - fd null -> using defaults\n" );
13156: #endif
13157: }
13158: else
13159: {
13160:     fseek( fd, 0L, SEEK_END );
13161:     file_size = (size_t) ftell( fd );
13162:     fseek( fd, 0L, SEEK_SET );
13163:     if ( file_size != sizeof( struct idi_dataset ) )
13164:     {
13165:         IDI_Dataset_Defaults( dataset );
13166: #if defined( MAIN_DEBUG )
13167: printf( "IDI_Initialization_Data_Structure_Load - file_size=%u\n" );
13168: # if defined( __MSDOS__ )
13169:     printf( "%u, sizeof(struct idi_dataset)=%u\n", file_size, sizeof( struct idi_dataset )
13170: );
13170: # else
13171:     printf( "%lu, sizeof(struct idi_dataset)=%lu\n", file_size, sizeof( struct
idi_dataset ) );
13172: # endif
13173: #endif
13174:     }
13175:     else
13176:     {
13177:         fread( dataset, sizeof( struct idi_dataset ), 1, fd );
13178: #if defined( MAIN_DEBUG )
13179: printf( "IDI_Initialization_Data_Structure_Load - file_size=%u\n" );
13180: # if defined( __MSDOS__ )
13181:     printf( "%u\n", file_size );
13182: # else
13183:     printf( "%lu\n", file_size );
13184: # endif
13185: #endif
13186:     }
13187:     fclose( fd );
13188: }
13189: dataset->svn_revision_string      = svn_revision_string;
13190: dataset->irq_handler_active      = false;
13191: dataset->irq.PleaseInstallHandlerRequest = false;
13192: dataset->irq_count              = 0;
13193: dataset->irq_count_previous    = 0;
13194: dataset->spi_is_present        = false;
13195:
13196: dataset->bank_previous          = IDI_BANK_UNDEFINED;
13197: dataset->quit_application      = false;
13198: dataset->board_id              = ID_IDI48;
13199: }
13200:
13205: static void IDO_Initialization_Data_Structure_Load( struct ido_dataset * dataset )
13206: {
13207:     FILE * fd;
13208:     size_t file_size;
13209:
13210:     fd = fopen( IDO_BOARD_INIT_FILE_NAME, "rb" );
13211:     if ( NULL == fd )
13212:     {
```

```

13213:     IDO_Dataset_Defaults( dataset );
13214: #if defined( MAIN_DEBUG )
13215: printf( "IDO_Initialization_Data_Structure_Load - fd null -> using defaults\n" );
13216: #endif
13217: }
13218: else
13219: {
13220:     fseek( fd, 0L, SEEK_END );
13221:     file_size = (size_t) ftell( fd );
13222:     fseek( fd, 0L, SEEK_SET );
13223:     if ( file_size != sizeof( struct ido_dataset ) )
13224:     {
13225:         IDO_Dataset_Defaults( dataset );
13226: #if defined( MAIN_DEBUG )
13227: printf( "IDO_Initialization_Data_Structure_Load - file_size=%u\n" );
13228: # if defined( __MSDOS__ )
13229:     printf( "%u, sizeof(struct ido_dataset)=%u\n", file_size, sizeof( struct ido_dataset )
13230: );
13231: # else
13231:     printf( "%lu, sizeof(struct ido_dataset)=%lu\n", file_size, sizeof( struct
ido_dataset ) );
13232: # endif
13233: #endif
13234: }
13235: else
13236: {
13237:     fread( dataset, sizeof( struct ido_dataset ), 1, fd );
13238: #if defined( MAIN_DEBUG )
13239: printf( "IDO_Initialization_Data_Structure_Load - file_size=%u\n" );
13240: # if defined( __MSDOS__ )
13241:     printf( "%u\n", file_size );
13242: # else
13243:     printf( "%lu\n", file_size );
13244: # endif
13245: #endif
13246: }
13247: fclose( fd );
13248: }
13249: dataset->svn_revision_string      = svn_revision_string;
13250: dataset->irq_handler_active      = false;
13251: dataset->irq.Please_install_handler_request = false;
13252: dataset->irq_count              = 0;
13253: dataset->irq_count_previous     = 0;
13254: dataset->spi_is_present        = false;
13255:
13256: dataset->bank_previous          = IDO_BANK_UNDEFINED;
13257: dataset->quit_application       = false;
13258: dataset->board_id               = ID_IDO48;
13259: }
13260:
13265: static void IDI_Initialization_Register( int board_id, struct idi_dataset * dataset )
13266: {
13267:     int index;
13268:
13269:     for ( index = 0; index < REGS_INIT_QTY; index++ )
13270:     {
13271:         if ( dataset->reg_init[index].used )
13272:         {
13273:             if ( IDI_UNDEFINED != dataset->reg_init[index].location )
13274:             {
13275:                 IO_Write_U8( board_id, __LINE__, dataset->reg_init[index].location,
dataset->reg_init[index].value );
13276:             }
13277:         else

```

```
13278:     {
13279:         IO_Write_U8_Port( (struct board_dataset *) dataset,
13280:                             __LINE__,
13281:                             dataset->reg_init[index].address,
13282:                             dataset->reg_init[index].value
13283:             );
13284:     }
13285: }
13286: }
13287: }
13288:
13293: static void IDO_Initialization_Register( int board_id, struct ido_dataset * dataset )
13294: {
13295:     int index;
13296:
13297:     for ( index = 0; index < REGS_INIT_QTY; index++ )
13298:     {
13299:         if ( dataset->reg_init[index].used )
13300:         {
13301:             if ( IDO_UNDEFINED != dataset->reg_init[index].location )
13302:             {
13303:                 IO_Write_U8( board_id, __LINE__, dataset->reg_init[index].location,
dataset->reg_init[index].value );
13304:             }
13305:         else
13306:         {
13307:             IO_Write_U8_Port( (struct board_dataset *) dataset,
13308:                               __LINE__,
13309:                               dataset->reg_init[index].address,
13310:                               dataset->reg_init[index].value
13311:             );
13312:         }
13313:     }
13314: }
13315: }
13316:
13317:
13327: int Initialization( int board_id )
13328: {
13329:     struct board_dataset * dataset;
13330:     (void) board_id;
13331:
13332:     dataset = (struct board_dataset *) board_definition[ID_IDI48].dataset;
13333:     dataset->quit_application = false;
13334:     IDI_Initialization_Data_Structure_Load( (struct idi_dataset *) dataset );
13335:     dataset = (struct board_dataset *) board_definition[ID_IDO48].dataset;
13336:     dataset->quit_application = false;
13337:     IDO_Initialization_Data_Structure_Load( (struct ido_dataset *) dataset );
13338:
13339: #if defined( __MSDOS__ )
13340:     IOKern_Resource_Initialization();
13341: #endif
13342:     dataset = (struct board_dataset *) board_definition[ID_IDI48].dataset;
13343:     IDI_Initialization_Register( ID_IDI48, (struct idi_dataset *) dataset );
13344:     dataset = (struct board_dataset *) board_definition[ID_IDO48].dataset;
13345:     IDO_Initialization_Register( ID_IDO48, (struct ido_dataset *) dataset );
13346:
13347:
13348:     signal( SIGINT, Signal_Handler );
13349:
13350:     IO_Trace_Initialize( ID_IDI48 );
13351:     IO_Trace_Initialize( ID_IDO48 );
13352:
13353: #if(0)
```

```
13354: struct board_dataset * dataset;
13355: dataset = ( struct board_dataset * ) board_definition[board_id].dataset;
13356:
13357:
13358: switch( board_id )
13359: {
13360:     case ID_IDI48:
13361:         IDI_Initialization_Data_Structure_Load( (struct idi_dataset *) dataset );
13362:         break;
13363:     case ID_IDO48:
13364:         IDO_Initialization_Data_Structure_Load( (struct ido_dataset *) dataset );
13365:         break;
13366:     default:
13367:         return -EC_BOARD_TYPE;
13368: }
13369: #if defined( __MSDOS__ )
13370: IOKern_Resource_Initialization();
13371: #endif
13372:
13373: switch( board_id )
13374: {
13375:     case ID_IDI48:
13376:         IDI_Initialization_Register( board_id, (struct idi_dataset *) dataset );
13377:         break;
13378:     case ID_IDO48:
13379:         IDO_Initialization_Register( board_id, (struct ido_dataset *) dataset );
13380:         break;
13381:     default:
13382:         return -EC_BOARD_TYPE;
13383: }
13384:
13385:
13386: dataset->quit_application = false;
13387: signal( SIGINT, Signal_Handler );
13388: #endif
13389:     return SUCCESS;
13390: }
13391:
13392:
13393: int IDI_Main_Loop_Testing( struct idi_dataset * dataset )
13394: {
13395:     if ( dataset->irq_handler_active )
13396:     {
13397:         if ( dataset->irq_count != dataset->irq_count_previous )
13398:         {
13399:             dataset->irq_count_previous = dataset->irq_count;
13400:
13401:             if ( dataset->irq_handler_active )
13402:                 printf( " --> irq %u: count = %d\n", dataset->irq_number, dataset->irq_count );
13403:             else
13404:                 printf( " --> irq %u: count = %lu\n", dataset->irq_number, dataset->irq_count );
13405:         }
13406:         if ( dataset->irq_count_previous >= dataset->irq_quantity )
13407:             return -1;
13408:     }
13409: }
13410:     if ( dataset->irq_count_previous >= dataset->irq_quantity )
13411:     {
13412:         return -1;
13413:     }
13414: }
13415: }
13416: return 0;
13417: }
13418:
13419: static void Main_Versalogic( void )
13420: {
13421:
13422:
```

```
13424:  
13425: #if defined( __MSDOS__ )  
13426:  
13427:     outportb( 0x00E2, 2 );  
13428: #endif  
13429: }  
13430:  
13431:  
13443: int main( int argc, char* argv[] )  
13444: {  
13445:  
13446:  
13447:     int board_id;  
13448:     int error_code;  
13449:     int index;  
13450:     int argc_new;  
13451:     char ** argv_new;  
13452:     struct board_dataset * dataset;  
13453:  
13454:  
13455: #if defined( MAIN_DEBUG )  
13456: printf( "Main: Entry\n" );  
13457: #endif  
13458: #if defined( SPI_PRINT_XFR_DEBUG )  
13459: global_internal_tx_byte_count = 0;  
13460: global_internal_rx_byte_count = 0;  
13461: #endif  
13462:  
13463:  
13464:  
13465:  
13466: # if(1)  
13467:     setvbuf(stdout, NULL, _IONBF, 0);  
13468:     setvbuf(stderr, NULL, _IONBF, 0);  
13469: # endif  
13470:  
13471:  
13472: #if defined( MAIN_DEBUG )  
13473: printf( "Main: 1\n" );  
13474: #endif  
13475:  
13476: #if(0)  
13477: {  
13478:     char buf[80];  
13479:     int count = argc;  
13480:     int index = 0;  
13481:     while ( count > 0 )  
13482:     {  
13483:         printf( "index = %d, str = <%s>\n", index, argv[index] );  
13484:         index++;  
13485:         count--;  
13486:     }  
13487:     Time_Current_String( buf, 80 );  
13488:     printf( "Current time is: %s", buf );  
13489: }  
13490: #endif  
13491:  
13492:  
13493:     Main_Versalogic();  
13494:  
13495: #if defined( MAIN_DEBUG )  
13496: printf( "Main: 2\n" );  
13497: #endif  
13498:
```

```
13499:     error_code      = SUCCESS;
13500:     board_id        = ID_BOARD_UNKNOWN;
13501:     global_board_id = board_id;
13502:
13503:     global_loop_space = false;
13504:
13505:     global_io_trace_enable = false;
13506:
13507:     index = 1;
13508:     if ( argc > 1 )
13509:     {
13510:
13511:
13512:
13513:
13514:
13515:
13516: #if defined( MAIN_DEBUG )
13517: printf( "Main: 3\n" );
13518: #endif
13519:
13520:
13521:     argv_new    = &(argv[index]);
13522:     argc_new    = argc - index;
13523:     error_code = Command_Line_Prefix( argc_new, argv_new );
13524:     if ( error_code < 0 ) goto Main_Termination_Skip_Termination;
13525:     else
13526:         index += error_code;
13527:
13528:
13529:
13530:     switch( Command_Line_Main_Board_Type_Status( argv[index], &board_id ) )
13531:     {
13532:         case CLM_BTS_NORMAL:
13533:             break;
13534:         case CLM_BTS_IMPLIED:
13535:             global_board_id = board_id;
13536:             goto Main_Initialization_Start;
13537:         case CLM_BTS_NOT_REQUIRED:
13538:             goto Main_Loop_Start;
13539:     }
13540:
13541:
13542:     if ( 0 == strcmpi( "idi", argv[index] ) )
13543:     {
13544:         board_id    = ID_IDI48;
13545:         global_board_id = board_id;
13546:         index++;
13547:     }
13548:     else if ( 0 == strcmpi( "ido", argv[index] ) )
13549:     {
13550:         board_id    = ID_IDO48;
13551:         global_board_id = board_id;
13552:         index++;
13553:     }
13554:     else
13555:     {
13556:         error_code = -EC_BOARD_TYPE;
13557:         goto Main_Termination_Error_Codes_Skip_Termination;
13558:     }
13559:
13560:
13561: Main_Initialization_Start:
13562: #if defined( MAIN_DEBUG )
```

```
13563: printf( "Main: Main_Initialization_Start\n" );
13564: #endif
13565:         error_code = Initialization( board_id );
13566:         if ( SUCCESS != error_code ) goto Main_Termination;
13567:
13568:         dataset      = ( struct board_dataset * ) board_definition[board_id].dataset;
13569:
13570:
13571: Main_Loop_Start:
13572: #if defined( MAIN_DEBUG )
13573: printf( "Main: Main_Loop_Start\n" );
13574: #endif
13575:     do
13576:     {
13577:         argv_new = &(argv[index]);
13578:         argc_new = argc - index;
13579:         error_code = Command_Line_Main( board_id, argc_new, argv_new );
13580:         if ( SUCCESS != error_code ) goto Main_Termination_Error_Codes;
13581:
13582:         switch( board_id )
13583:         {
13584:             case ID_IDI48:
13585:                 if ( -1 == IDI_Main_Loop_Testing( (struct idi_dataset *) dataset ) )
13586:                 {
13587:                     goto Main_Termination;
13588:                 }
13589:                 break;
13590:         }
13591:
13592:         if ( global_loop_command )
13593:         {
13594:             if ( global_loop_delay_ms > 0 )
13595:             {
13596: #if defined( __MSDOS__ )
13597:                 delay( global_loop_delay_ms );
13598: #endif
13599:             }
13600:
13601:             if ( global_loop_count > 0 )
13602:             {
13603:                 global_loop_count_counter--;
13604:                 if ( 0 == global_loop_count_counter ) global_loop_command = false;
13605:             }
13606:
13607:             if ( true == global_loop_space )
13608:             {
13609:                 int character;
13610:                 BOOL not_done;
13611:
13612:                 not_done = true;
13613:                 do
13614:                 {
13615:                     if ( Character_Get( &character ) )
13616:                     {
13617:                         if ( ' ' == ( (char) character ) )
13618:                         {
13619:                             not_done = false;
13620:                         }
13621:                         else
13622:                         {
13623:                             not_done          = false;
13624:                             global_loop_command = false;
13625:                         }
13626:                 }
```

```

13627:     } while( true == not_done );
13628: }
13629: else
13630: {
13631:     if ( Character_Get( NULL ) ) global_loop_command = false;
13632: }
13633:
13634:
13635: }
13636:
13637: } while ( global_loop_command );
13638: }
13639: else
13640: {
13641:     Help( false , stdout );
13642:     goto Main_Termination_Skip_Termination;
13643: }
13644:
13645: Main_Termination:
13646:
13647: #if defined( MAIN_DEBUG )
13648: printf( "Main: Main_Termination\n" );
13649: #endif
13650:     Termination( board_id );
13651:
13652: Main_Termination_Skip_Termination:
13653:
13654: #if defined( MAIN_DEBUG )
13655: printf( "Main: Main_Termination_Skip_Termination\n" );
13656: #endif
13657:
13658:     return error_code;
13659:
13660: Main_Termination_Error_Codes:
13661:
13662:     Termination( board_id );
13663:
13664: Main_Termination_Error_Codes_Skip_Termination:
13665:     printf( "ERROR: %d, %s\n", error_code, EC_Code_To_Human_Readable( error_code ) );
13666:     return error_code;
13667: }
```

Enumerations

	Name	Description
☞	buffer_column (☞ see page 299)	brief
☞	CLM_BTS (☞ see page 299)	brief
☞	IO_TRACE_ENUM (☞ see page 306)	
☞	IO_TRACE_START_ENUM (☞ see page 307)	
☞	IO_TRACE_STOP_ENUM (☞ see page 307)	

	irqreturn (see page 308)	<p>The following code is directly from irqreturn.h from the Linux kernel tree. This is being used by the simulator and DOS to emulate Linux driver behavior and specifically resource allocations.</p> <pre>* • enum irqreturn • @IRQ_NONE interrupt was not from this device • @IRQ_HANDLED interrupt was handled by this device • @IRQ_WAKE_THREAD handler requests to wake the handler thread</pre>
	BANK_ENUM (see page 312)	This is type BANK_ENUM.
	EC_ENUM (see page 313)	EC = Error Code
	IDI_BANK_ENUM (see page 313)	This is type IDI_BANK_ENUM.
	IDI_REG_ENUM (see page 313)	brief Once the register location (i.e. logical_address or row) is known, then all other parameters associated with that register can be looked up including physical_offset, bank and legal read/write access.
	IDO_BANK_ENUM (see page 314)	This is type IDO_BANK_ENUM.
	IDO_REG_ENUM (see page 314)	
	IOKERN_CHAIN_TO_OLD_ENUM (see page 314)	
	IOKERN_IRQ_ENUM (see page 315)	
	IOKERN_TASK_ID_ENUM (see page 316)	
	MODE_JUMPERS_ENUM (see page 316)	
	REG_DIR_ENUM (see page 317)	
	SPI_CS_B_ENUM (see page 317)	
	TEST_STATE_ENUM (see page 317)	

Legend

	Enumeration
---	-------------

Functions

	Name	Description
	ADS1259_Calibration_Range_Test (see page 28)	brief
	AS_Calibration_Value_Read (see page 29)	brief
	AS_Calibration_Value_Write (see page 30)	brief
	AS_Chip_Select_Route_Override (see page 31)	brief Analog Stick SPI chip select over-ride mechanism. Temporarily used to set the chip select channel to what we need based on "set frams".
	AS_Chip_Select_Route_Restore (see page 31)	brief Analog Stick SPI chip select restore mechanism. Returns the chip select to original SPI configuration settings.
	AS_Data_Read (see page 32)	brief Rdata command
	AS_Data_Ready_NoWait (see page 33)	brief
	AS_Data_Ready_Wait (see page 33)	brief

AS_Opcode_Write (see page 34)	brief
AS_Rdata_Statistics (see page 35)	brief
AS_Register_Defaults (see page 36)	brief
AS_Register_Name_To_Offset (see page 36)	brief
AS_Registers_Read (see page 37)	brief
AS_Registers_Write (see page 38)	brief
AS_Time_Statistics (see page 39)	
Bank_Name_To_Symbol (see page 40)	This is function Bank_Name_To_Symbol.
Bank_Symbol_To_Name (see page 41)	This is function Bank_Symbol_To_Name.
Bit_String_Index_By_Name (see page 41)	brief
Bit_String_Report (see page 42)	brief
Buffer_Init (see page 43)	brief
Buffer_Length (see page 43)	brief
Buffer_Maximum_Int32 (see page 44)	brief
Buffer_Mean_Int32 (see page 44)	brief
Buffer_Minimum_Int32 (see page 45)	brief
Buffer_Peak_To_Peak_Int32 (see page 45)	brief
Buffer_Save (see page 46)	brief
Buffer_Save_Binary_Int32 (see page 46)	brief
Buffer_Standard_Deviation_Int32 (see page 47)	Uses Bessel's correction.
Buffer_Stuff (see page 48)	brief
Character_Get (see page 48)	brief Obtains a key from the keyboard in a non-blocking way. This is exerpetted from AES Universal Library/Driver. Excerpt from AES advanced Linux library/driver. COPYRIGHT NOTICE Copyright (c) 2012 by Apex Embedded Systems. param[out] character Character from keyboard otherwise null character.
CMD_AS_Calibration_Gain (see page 49)	brief
CMD_AS_Calibration_Offset (see page 50)	brief
CMD_AS_Gancal (see page 51)	brief
CMD_AS_Ofscal (see page 51)	brief
CMD_AS_Parameter (see page 52)	brief
CMD_AS_Rdata (see page 54)	brief In order to use Data Read in Stop Continuous Mode one must issue the following command sequence: b idi spi mode 1 b idi spi clk 100000 b idi as sdatac b idi as wreg 0 0x85 0x90 0x00 0x00 0x00 0x00 0x00 0x40 b idi as rreg 0 9 b idi as start b idi as rdata 1024 --file out.txt
CMD_AS_Rdatac (see page 58)	brief
CMD_AS_Reg_Write_By_Name (see page 58)	brief
CMD_AS_Register_Load (see page 60)	brief
CMD_AS_Register_Save (see page 61)	brief
CMD_AS_Reset (see page 61)	brief
CMD_AS_RReg (see page 62)	brief
CMD_AS_Sdatac (see page 64)	brief
CMD_AS_Sleep (see page 64)	brief
CMD_AS_Start (see page 65)	brief

≡	CMD__AS_Stop (█ see page 65)	brief
≡	CMD__AS_Wakeup (█ see page 66)	brief
≡	CMD__AS_WReg (█ see page 66)	brief
≡	CMD__FRAM_Dump (█ see page 67)	brief
≡	CMD__FRAM_Init (█ see page 68)	brief
≡	CMD__FRAM_Load (█ see page 69)	brief
≡	CMD__FRAM_RDID (█ see page 70)	brief
≡	CMD__FRAM_RDSR (█ see page 70)	brief
≡	CMD__FRAM_Save (█ see page 71)	brief
≡	CMD__FRAM_WRDI (█ see page 72)	brief
≡	CMD__FRAM_WREN (█ see page 72)	brief
≡	CMD__FRAM_Write (█ see page 73)	brief
≡	CMD__FRAM_WRSR (█ see page 73)	brief
≡	CMD__IAI16_AI_All (█ see page 74)	brief
≡	CMD__IAI16_AI_Channel (█ see page 75)	brief
≡	CMD__IAI16_AI_ID (█ see page 75)	brief
≡	CMD__IDI48_DIN_All (█ see page 76)	brief
≡	CMD__IDI48_DIN_Channel (█ see page 77)	brief
≡	CMD__IDI48_DIN_Group (█ see page 78)	brief
≡	CMD__IDI48_DIN_Helper_All_Values_False (█ see page 79)	brief
≡	CMD__IDI48_DIN_ID (█ see page 79)	brief
≡	CMD__IDI48_DIN_Test (█ see page 80)	brief
≡	CMD__IDI48_DIN_Test_FE (█ see page 81)	brief
≡	CMD__IDI48_DIN_Test_Interrupt (█ see page 84)	brief Assumption: IRQs slower than main (█ see page 239) loop below.
≡	CMD__IDI48_DIN_Test_Interrupt_Handler (█ see page 89)	brief Interrupt Service Routine (ISR). This is a legacy implementation.
≡	CMD__IDI48_DIN_Test_RE (█ see page 90)	brief
≡	CMD__IDI48_DIN_Test_Value (█ see page 93)	brief
≡	CMD__IDO48_DO_All (█ see page 96)	This is function CMD__IDO48_DO_All.
≡	CMD__IDO48_DO_Channel (█ see page 97)	brief
≡	CMD__IDO48_DO_Group (█ see page 98)	brief
≡	CMD__IDO48_DO_ID (█ see page 99)	brief
≡	CMD__IDO48_DOUT_Test_Alternate (█ see page 100)	brief
≡	CMD__IDO48_DOUT_Test_One_Hot (█ see page 102)	brief
≡	CMD__IDO48_IDI48_Loopback (█ see page 103)	brief
≡	CMD__IDO48_Test (█ see page 109)	brief Cycles through all channels one at a time in a one-hot situation.
≡	CMD__Main_AnalogStick_CS (█ see page 110)	brief
≡	CMD__Main_Base (█ see page 110)	brief
≡	CMD__Main_FRAM_CS (█ see page 111)	brief

⌘	CMD__Main_I_Count (see page 112)	brief Number of interrupt cycles. If "loop" is used, then the count can be terminated by pressing any key on the keyboard input.
⌘	CMD__Main_Init_Reg (see page 113)	brief
⌘	CMD__Main_IO_Behavior (see page 114)	brief
⌘	CMD__Main_Irq_Number (see page 115)	brief
⌘	CMD__Main_Mode_Jumpers (see page 116)	brief Mode jumper settings. 0 = 00 =
⌘	CMD__Main_Trace (see page 117)	brief
⌘	CMD__SPI_Commit (see page 118)	brief
⌘	CMD__SPI_Config_Chip_Select_Behavior (see page 118)	brief
⌘	CMD__SPI_Config_Chip_Select_Route (see page 120)	brief
⌘	CMD__SPI_Config_Clock_Hz (see page 120)	brief
⌘	CMD__SPI_Config_End_Cycle_Delay_Sec (see page 121)	brief
⌘	CMD__SPI_Config_Get (see page 122)	brief
⌘	CMD__SPI_Config_Mode (see page 123)	CPOL CPHA MODE 0 0 0 1 0 1 2 1 0 3 1 1
⌘	CMD__SPI_Config_SDI_Polarity (see page 124)	brief
⌘	CMD__SPI_Config_SDIO_Wrap (see page 125)	brief
⌘	CMD__SPI_Config_SDO_Polarity (see page 126)	brief
⌘	CMD__SPI_Data (see page 126)	brief
⌘	CMD__SPI_Data_Interpreter (see page 128)	brief
⌘	CMD__SPI_FIFO (see page 129)	brief
⌘	CMD__SPI_ID (see page 132)	brief
⌘	CMD__SPI_Status (see page 132)	brief
⌘	CMD__Trace_File (see page 133)	brief
⌘	CMD__Trace_Start (see page 134)	brief
⌘	CMD__Trace_Stop (see page 135)	brief
⌘	Command_Line_Analog_Stick (see page 136)	brief
⌘	Command_Line_Dump (see page 137)	brief
⌘	Command_Line_FPGA (see page 138)	brief
⌘	Command_Line_FRAM (see page 139)	brief
⌘	Command_Line_Help (see page 140)	brief
⌘	Command_Line_IAI16 (see page 140)	brief
⌘	Command_Line_IDI48 (see page 141)	brief
⌘	Command_Line_IDO48 (see page 142)	brief
⌘	Command_Line_IO_Read (see page 143)	brief used to read one or more values from registers sequentially. b [prefix commands] [x] [c]
⌘	Command_Line_IO_Write (see page 145)	brief Used to write to one or more registers in sequence. Can write values to the same register by using the 'x' to toggle auto-indexing from initially on to the off position.
⌘	Command_Line_Loop_Count (see page 146)	brief

.Command_Line_Loop_Delay (see page 147)	brief
.Command_Line_Loop_Space (see page 148)	brief
.Command_Line_Main (see page 148)	brief Processes and dispatches the top level of the command and passes the remaining string list onto specialized functions to further process arguments. If no command is specified then a help output is produced. param[in] argc number of arguments including the executable file name param[in] argv list of string arguments lex'd from the command line
.Command_Line_Main__Board_Type_Status (see page 149)	brief Determine if this command requires specification of board type or not, or if a board type can be implied.
.Command_Line_Prefix (see page 150)	brief
.Command_Line_Prefix_Irq (see page 151)	brief
.Command_Line_Prefix_Loop (see page 152)	brief
.Command_Line_Prefix_Trace (see page 153)	brief
.Command_Line_Register_Transaction (see page 154)	brief Either reads or writes a register using the form: id1 [] If is not include, then it is assumed to be a read. If value is included, then a write to the specified register is made. This function uses the definitions[] array which is global and built from IDI_REGISTER_SET_DEFINITION (see page 350) macro which is a nicely organized register list. param[in] argc number of arguments including the executable file name param[in] argv list of string arguments lex'd from the command line
.Command_Line_Set (see page 156)	brief
.Command_Line_SPI (see page 157)	brief
.Command_Line_Wait (see page 158)	brief
.EC_Code_To_Human_Readable (see page 158)	This is function EC_Code_To_Human_Readable.
.FPGA_Date_Time_Information (see page 159)	brief
.FRAM__Chip_Select_Route_Override (see page 161)	brief FRAM SPI chip select over-ride mechanism. Temporarily used to set the chip select channel to what we need based on "set frams".
.FRAM__Chip_Select_Route_Restore (see page 162)	brief FRAM SPI chip select restore mechanism. Returns the chip select to original SPI configuration settings.
.FRAM__Memory_Read (see page 162)	brief Reads data from FRAM memory to the output buffer (see page 325). param[in] address Starting FRAM address param[in] count Number of bytes to transfer param[out] buffer (see page 325) Destination buffer (see page 325) in which to store the data
.FRAM__Memory_Write (see page 164)	brief Writes data from buffer (see page 325) to FRAM memory. param[in] address Starting FRAM address param[in] count Number of bytes to transfer param[out] buffer (see page 325) Source buffer (see page 325) from which data will be transferred to FRAM

FRAM__Read_ID (see page 165)	brief param[out] id The 32-bit ID register read from the FRAM. This appears to be only available with Fujitsu parts.
FRAM__Read_Status_Register (see page 166)	brief Read the FRAM status register and output the value.
FRAM__Write_Disable (see page 167)	brief FRAM Write Latch disable (or clear) command (WRDI).
FRAM__Write_Enable_Latch_Set (see page 168)	brief FRAM Write Enable Latch Set command (WREN)
FRAM__Write_Status_Register (see page 169)	brief Write to the FRAM status register. param[in] FRAM status value to be written.
FRAM_File_To_Memory (see page 170)	brief
FRAM_Memory_To_File (see page 171)	brief
FRAM_Report (see page 172)	brief
FRAM_Set (see page 173)	This function will be used when creating a memory pool so that as blocks are allocated one can determine if we have an issue outside of any allocated space (i.e. overflows and so on). param[in] cfg pass in the configuration to be written to hardware. return a nonzero if successful, else return zero.
Help (see page 175)	brief Outputs a help listing to the user.
Help_Output (see page 176)	brief
Help_Pause_Helper (see page 177)	brief Outputs a help listing to the user.
Hex_Dump_Line (see page 178)	This is function Hex_Dump_Line.
IAI_AI_ID_Get (see page 179)	brief Obtains the analog input component (or board ID in this case) ID number. param[out] id The 16-bit ID number
IAI16_AI_Channel_Get (see page 179)	brief
IAI16_AI_Channel_Multiple_Get (see page 180)	brief
IDI_Bank_Name_To_Symbol (see page 181)	brief param[in] bank the bank enumerated value written to the bank register.
IDI_Bank_Symbol_To_Name (see page 181)	brief Outputs a human readable CSV to the desired output file or stdout. param[in] bank the bank enumerated value written to the bank register.
IDI_Dataset_Defaults (see page 182)	brief
IDI_DIN_Channel_Get (see page 183)	brief Obtains and reports a single digital input channel. param[in] channel channel to be read out. param[out] value Pointer to the boolean value to be set based on the digital input value
IDI_DIN_Group_Get (see page 183)	brief Reads the selected digital input port (8-bits). param[in] group the group, range is 0 to 5. param[out] value pointer to the destination for the data read out
IDI_DIN_ID_Get (see page 184)	brief Obtains the digital input component (or board ID in this case) ID number. param[out] id The 16-bit ID number
IDI_DIN_IsNotPresent (see page 185)	brief Determines if the DIN component and/or board is present. Returns true if not present (i.e. error).

IDI_DIN_Pending_Clear (see page 185)	brief param[in] channel channel to be cleared of its pending status. param[out] value pointer to the destination for the data read out
IDI_Help (see page 186)	brief
IDI_Initialization_Data_Structure_Load (see page 186)	brief
IDI_Initialization_Register (see page 187)	brief
IDI_Main_Loop_Testing (see page 188)	brief
IDI_Register_Report_CSV (see page 189)	brief
IDI_Termination (see page 190)	brief
IDI48_DIN_ID_Port_Scan (see page 191)	brief
IDO_Bank_Name_To_Symbol (see page 192)	This is function IDO_Bank_Name_To_Symbol.
IDO_Bank_Symbol_To_Name (see page 193)	This is function IDO_Bank_Symbol_To_Name.
IDO_Dataset_Defaults (see page 193)	brief
IDO_DO_Channel_Get (see page 194)	brief Obtains and reports a single digital output channel. param[in] channel channel to be read out. param[out] value Pointer to the boolean value to be set based on the digital input value
IDO_DO_Channel_Set (see page 195)	brief Sets a single digital output channel. param[in] channel to be written. param[in] value
IDO_DO_Group_Get (see page 195)	brief Reads the selected digital output port (8-bits). param[in] group the group, range is 0 to 5. param[out] value pointer to the destination for the data read out
IDO_DO_Group_Set (see page 196)	brief Reads the selected digital output port (8-bits). param[in] group the group, range is 0 to 5. param[in] value pointer to the destination for the data read out
IDO_DOUT_ID_Get (see page 196)	brief Obtains the digital output component (or board ID in this case) ID number. param[out] id The 16-bit ID number
IDO_DOUT_IsNotPresent (see page 197)	brief Determines if the DIN component and/or board is present. Returns true if not present (i.e. error).
IDO_Help (see page 198)	brief
IDO_Initialization_Data_Structure_Load (see page 198)	brief
IDO_Initialization_Register (see page 199)	brief
IDO_Register_Report_CSV (see page 200)	brief
IDO_Termination (see page 201)	brief
IDO48_IDI48_Loopback_Test (see page 202)	brief
Initialization (see page 202)	brief Runs upon application startup. It restores the idi_dataset (see page 302) data structure or if the file cannot be found it will simply initialize those parameters to default values.
IO_Direction_IsNotValid (see page 204)	brief Looks up in the register definitions list for the ports possible read/write directions. param[in] location the enumerated register symbol. The enumerated symbol is composed of offset and bank information used to determine the final address information. param[in] direction The desired direction that is to run

✳️	IO_Get_Symbol_Name (see page 204)	brief Translates a register enumerated symbol into a string that is the same as the enumerated symbol used throughout this code base. param[in] location The enumerated symbol representing the register.
✳️	IO_Read_U16_Address_Fixed (see page 205)	brief Reads uint16_t (see page 321) from I/O ports in a uint8_t (see page 321) succession to the same address location. Macros are used to guide the target implementation. In this case, bus width (which we typically refer to the port width, which is different than register width) is assumed to be byte (uint8_t (see page 321) wide). param[in] location the enumerated register symbol. The enumerated symbol is composed of offset and bank information used to determine the final address information. param[in] value The pointer to the destination of the read uint16_t (see page 321) value.
✳️	IO_Read_U16_Address_Increment (see page 206)	brief Reads uint16_t (see page 321) from I/O ports in a uint8_t (see page 321) succession incrementing the offset address. Macros are used to guide the target implementation. In this case, bus width (which we typically refer to the port width, which is different than register width) is assumed to be byte (uint8_t (see page 321) wide). param[in] location the enumerated register symbol. The enumerated symbol is composed of offset and bank information used to determine the final address information. param[in] value The pointer to the destination of the read uint16_t (see page 321) value.
✳️	IO_Read_U8 (see page 206)	brief Reads uint8_t (see page 321) from I/O port. Macros are used to guide the target implementation. param[in] location the enumerated register symbol. The enumerated symbol is composed of offset and bank information used to determine the final address information. param[in] value The pointer to the destination of the read uint8_t (see page 321) value.
✳️	IO_Read_U8_Port (see page 208)	brief
✳️	IO_Trace_Dump (see page 208)	brief Dumps the trace buffer (see page 325) to a CSV file for further analysis.
✳️	IO_Trace_Initialize (see page 211)	brief
✳️	IO_Trace_Log (see page 212)	brief
✳️	IO_Trace_Log_Dataset (see page 212)	brief
✳️	IO_Trace_Log_Dataset_Begin (see page 213)	brief
✳️	IO_Trace_Log_Dataset_End (see page 214)	brief
✳️	IO_Trace_Log_Function (see page 215)	brief

☰	IO_Write_U16_Address_Fixed (see page 216)	brief Writes uint16_t (see page 321) to I/O ports in a uint8_t (see page 321) succession to the same address location. Macros are used to guide the target implementation. In this case, bus width (which we typically refer to the port width, which is different than register width) is assumed to be byte (uint8_t (see page 321)) wide. param[in] location the enumerated register symbol. The enumerated symbol is composed of offset and bank information used to determine the final address information. param[in] value The uint16_t (see page 321) value to be written to the I/O register
☰	IO_Write_U16_Address_Increment (see page 216)	brief Writes uint16_t (see page 321) to I/O ports in a uint8_t (see page 321) succession incrementing the offset address. Macros are used to guide the target implementation. In this case, bus width (which we typically refer to the port width, which is different than register width) is assumed to be byte (uint8_t (see page 321)) wide. param[in] location the enumerated register symbol. The enumerated symbol is composed of offset and bank information used to determine the final address information. param[in] value The uint16_t (see page 321) value to be written to the I/O register
☰	IO_Write_U8 (see page 217)	brief Writes uint8_t (see page 321) to I/O port. Macros are used to guide the target implementation. param[in] board_id software level board selector or descriptor. param[in] location the enumerated register symbol. The enumerated symbol is composed of offset and bank information used to determine the final address information. param[in] value The data to be written out.
☰	IO_Write_U8_Port (see page 218)	brief
☰	IOKern_DOS_IRQ_Free (see page 219)	brief
☰	IOKern_DOS_IRQ_Mask_Get (see page 220)	
☰	IOKern_DOS_IRQ_Mask_Set (see page 220)	
☰	IOKern_DOS_IRQ_Request (see page 221)	brief
☰	IOKern_DOS_ISR_Install (see page 222)	brief
☰	IOKern_DOS_ISR_Restore (see page 223)	brief
☰	IOKern_DOS_Vector_Get (see page 224)	
☰	IOKern_DOS_Vector_Set (see page 225)	
☰	IOKern Interrupt_Helper (see page 225)	brief Supporting 'fast' interrupts
☰	IOKern_IRQ_Invoke_ISR_Test (see page 226)	brief
☰	IOKern_IRQ_Test (see page 226)	brief
☰	IOKern_IRQ_Test_Helper (see page 227)	brief
☰	IOKern_ISR0 (see page 228)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
☰	IOKern_ISR1 (see page 228)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
☰	IOKern_ISR10 (see page 229)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.

IOC	IOKern_ISR11 (see page 229)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
IOC	IOKern_ISR12 (see page 230)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
IOC	IOKern_ISR13 (see page 230)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
IOC	IOKern_ISR14 (see page 231)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
IOC	IOKern_ISR15 (see page 231)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
IOC	IOKern_ISR2 (see page 232)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
IOC	IOKern_ISR3 (see page 232)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
IOC	IOKern_ISR4 (see page 233)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
IOC	IOKern_ISR5 (see page 233)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
IOC	IOKern_ISR6 (see page 234)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
IOC	IOKern_ISR7 (see page 234)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
IOC	IOKern_ISR8 (see page 235)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
IOC	IOKern_ISR9 (see page 235)	brief Interrupt service routine. It will subsequently call the appropriate user function. These are meant to be short operations.
IOC	IOKern_PIC_EOI (see page 236)	brief
IOC	IOKern_Resource_Initialization (see page 236)	
IOC	IOKern_Resource_Termination (see page 237)	
IOC	IOKern_Task_Handle_Get (see page 238)	brief
IOC	IOKern_Task_ID_Get (see page 238)	brief
IOC	main (see page 239)	brief Processes and dispatches the top level of the command and passes the remaining string list onto specialized functions to further process arguments. If no command is specified then a help output is produced. param[in] argc number of arguments including the executable file name param[in] argv list of string arguments lex'd from the command line
IOC	Main_Versalogic (see page 243)	brief

≡	Print_Byte_List (see page 243)	brief
≡	Print_Multiple (see page 245)	brief
≡	Register_Acronym_To_Row (see page 245)	brief
≡	Register_Report_CSV (see page 246)	brief Outputs a human readable CSV to the desired output file or stdout. param[in] table register definition table or data structure array param[in] out destination of the human readable text to specified file or terminal.
≡	Signal_Handler (see page 247)	brief
≡	SPI_Calculate_Clock (see page 247)	brief Computes the SPI clock half clock register value given a requested SPI clock frequency. It will also produce a 'report' indicating the actual value (due to integer resolution) as well as a computed error between requested and actual. The error can be used to determine whether timing constraints are met. param[in] clock_request_hz Request clock frequency in Hertz. Example: 1.0e6 is 1MHz. param[in] clock_actual_hz Actual computed frequency. If this pointer is NULL, then it is not output. param[out] error Error between requested and actual. If this pointer is NULL, then it is not output. param[out] hci Half clock register value... more (see page 247)
≡	SPI_Calculate_End_Cycle_Delay (see page 248)	brief Computes the time delay at the end of each byte transmitted. It will only output the parameters whose pointers are not NULL. param[in] spi_half_clock_interval_sec Computed half clock interval in seconds param[in] delay_request_sec Requested time delay in seconds param[out] delay_actual_sec Pointer to actual time delay computed, if not NULL. param[out] error Pointer to error value computed, if not NULL. param[out] ecd Pointer to the computed end-cycle-delay, if not NULL.
≡	SPI_Calculate_Half_Clock (see page 249)	brief Computes the half clock register value given a requested time interval. It will also produce a 'report' indicating the actual value (due to integer resolution) as well as a computed error between requested and actual. The error can be used to determine whether timing constraints are met. param[in] half_clock_request_sec Request time interval in seconds. Example: 20.0e-6 is 20uS. param[in] half_clock_actual_sec Actual computed time. If this pointer is NULL, then it is not output. param[out] error Error between requested and actual. If this pointer is NULL, then it is not output. param[out] hci Half clock register value computed. If this... more (see page 249)
≡	SPI_Calculate_Half_Clock_Interval_Sec (see page 251)	brief Computes the half clock interval in seconds given the value from the half clock interval register. param[in] half_clock_interval Half clock interval register value
≡	SPI_Commit (see page 251)	brief Sets/Clears the chip select or used to commit the transmit/write FIFO to the spi interface. The mode of operation is dependent on the chip_select_behavior. param[in] commit Used to write to the SCS_COMMIT bit. Its behavior is dependent on the chip_select_behavior.
≡	SPI_Commit_Get (see page 252)	brief Sets/Clears the chip select or used to commit the transmit/write FIFO to the spi interface. The mode of operation is dependent on the chip_select_behavior. param[out] commit Used to write to the SCS_COMMIT bit. Its behavior is dependent on the chip_select_behavior.

≡	SPI_Commit_Is_Inactive (see page 253)	brief
≡	SPI_Configuration_Chip_Select_Behavior_Get (see page 253)	brief Extracts the chip select behavior from the SPI configuration register. param[out] chip_select_behavior pointer to the destination of the value obtained.
≡	SPI_Configuration_Chip_Select_Behavior_Set (see page 254)	brief Sets the chip select behavior to the SPI configuration register. param[in] chip_select_behavior enumerated value to be written to the register.
≡	SPI_Configuration_Chip_Select_Route_Get (see page 255)	brief Sets which hardware chip select line is to be used, either CS0 (channel 0) or CS1 (channel 1). param[out] chip_select_behavior pointer to the destination of the value obtained.
≡	SPI_Configuration_Chip_Select_Route_Set (see page 256)	brief Sets which hardware chip select line is to be used, either CS0 (channel 0) or CS1 (channel 1). param[in] cs_to_channel1 true if CS to route to channel 1 (CS1), otherwise channel 0 (CS0).
≡	SPI_Configuration_Get (see page 257)	brief Obtains the SPI configuration from the hardware. param[out] cfg SPI configuration data structure or data set
≡	SPI_Configuration_Initialize (see page 258)	brief Initializes the SPI configuration data structure param[in] cfg Pointer to the SPI configuration data structure to be initialized
≡	SPI_Configuration_Set (see page 259)	brief Commits the configuration data structure to the hardware. param[in] cfg The software configuration data structure to be committed to hardware
≡	SPI_Data_Read (see page 261)	Special case of Write/Read that has a function signature same as fread() or fwrite(). param[in] cfg pass in the configuration to be written to hardware. return a nonzero if successful, else return zero.
≡	SPI_Data_Write (see page 262)	Special case of Write/Read that has a function signature same as fread() or fwrite(). param[in] cfg pass in the configuration to be written to hardware. return a nonzero if successful, else return zero.
≡	SPI_Data_Write_Read (see page 263)	brief If CSB = software. Stuff as fast as you can. If CSB == buffer (see page 325). if object size = 1, then stuff as fast as you can. The chip select will go active and only go inactive when done transmitting. This is best used for objects whos size is less than SPI_FIFO_SIZE. if object size != 1, then stuff only that quantity and wait until buffers are empty in order to have the chip select wrap around the data. If CSB == uint8_t (see page 321) Stuff as fast as you can. If CSB == uint16_t (see page 321) Stuff two bytes at a time... more (see page 263)
≡	SPI_Data_Write_Read_Helper_Commit (see page 270)	brief returns true if user wishes to break.
≡	SPI_Data_Write_Read_Helper_Read (see page 271)	brief
≡	SPI_Data_Write_Read_Helper_Write (see page 273)	brief

✳️	SPI_FIFO_Read (🔗 see page 276)	<p>brief Reads from the SPI receive/read data FIFO. It does not attempt to correlate the number of transmit bytes with receive bytes. Its purpose is more for low level hardware testing. Note that this function has a signature identical to the fread() function (i.e. make use of function pointers to guide sourcing of data).</p> <p>param[in] buffer (🔗 see page 325) Buffer for the data destination. param[in] size Size of objects in bytes. param[in] count Number of objects to be read param[out] fd_log Optional log file to write the buffer (🔗 see page 325) too. If NULL, then no logging.</p>
✳️	SPI_FIFO_Write (🔗 see page 278)	<p>brief Writes specifically to the SPI transmit/write data FIFO. It does not attempt to correlate the number of transmit bytes with receive bytes. Its purpose is more for low level hardware testing. Note that this function has a signature identical to the fwrite() function (i.e. make use of function pointers to guide destination of data).</p> <p>param[in] buffer (🔗 see page 325) Buffer containing the data to be written. param[in] size Size of objects in bytes. param[in] count Number of objects to be written param[out] fd_log Optional log file to write the buffer (🔗 see page 325) too. If NULL, then no logging.</p>
✳️	SPI_ID_Get (🔗 see page 279)	This is function SPI_ID_Get.
✳️	SPI_ID_Get_Helper (🔗 see page 280)	<p>brief Retrieves the SPI ID register value.</p> <p>param[out] id The SPI component ID value.</p>
✳️	SPI_IsNotPresent (🔗 see page 280)	<p>brief Reports if the SPI component is available within the register space by matching a known ID. The SPI register map is only enabled within the hardware if the hardware mode is not zero (i.e. M1 and M0 jumpers on the board provide a nonzero value).</p>
✳️	SPI_Report_Configuration_Text (🔗 see page 281)	<p>brief Creates a human readable report of the SPI configuration data structure.</p> <p>param[in] cfg SPI configuration data structure pointer param[out] out File destination descriptor</p>
✳️	SPI_Report_Status_Text (🔗 see page 282)	<p>brief Produces a human readable report of the SPI status data structure.</p> <p>param[in] status SPI status data structure pointer param[out] out File destination descriptor</p>
✳️	SPI_Status_Read (🔗 see page 283)	<p>brief Builds a detailed status data structure of the receive/read incoming SPI data FIFO. Reports the quantity of bytes currently in the receive FIFO, full flag, empty flag, the total size of the FIFO in bytes, and sets tx_status to false indicating that this is status specific to the receive FIFO.</p> <p>The status register has the following format: status[7] full status[6] empty status[5] not used (future size expansion) status[4:0] number of bytes currently in the FIFO</p> <p>param[out] status Pointer to status data structure to be updated.</p>
✳️	SPI_Status_Read_FIFO_Is_Not_Empty (🔗 see page 284)	<p>brief Returns the receive/read FIFO empty status flag. This function is typically used to determine if the FIFO is empty.</p>

Method	SPI_Status_Read_FIFO_Status (see page 284)	brief Returns the complete read/receive FIFO status. The status register has the following format: status[7] full status[6] empty status[5] not used (future size expansion) status[4:0] number of bytes currently in the FIFO param[out] empty FIFO empty flag param[out] bytes_available a count of the number of bytes in the FIFO
Method	SPI_Status_Write (see page 286)	brief Builds a detailed status data structure of the transmit/write outgoing SPI data FIFO. Reports the quantity of bytes currently in the transmit FIFO, full flag, empty flag, the total size of the FIFO in bytes, and sets tx_status to true indicating that this is status specific to the transmit FIFO. The status register has the following format: status[7] full status[6] empty status[5] not used (future size expansion) status[4:0] number of bytes currently in the FIFO param[out] status Pointer to status data structure to be updated.
Method	SPI_Status_Write_FIFO_Is_Full (see page 287)	brief Returns the transmit/write FIFO full status flag. It is preferable to use the SPI_Status_Write (see page 286)() or SPI_Status_Write_FIFO_Status (see page 288)() because all status is retrieved at one time.
Method	SPI_Status_Write_FIFO_Is_Not_Empty (see page 288)	brief Returns the transmit/write FIFO empty status flag. This function is typically used to wait for the transmit/write FIFO to become empty.
Method	SPI_Status_Write_FIFO_Status (see page 288)	brief Returns the complete write/transmit FIFO status. The status register has the following format: status[7] full status[6] empty status[5] not used (future size expansion) status[4:0] number of bytes currently in the FIFO param[out] full FIFO full flag param[out] empty FIFO empty flag param[out] bytes_in_fifo a count of the number of bytes in the FIFO
Method	String_To_Bool (see page 293)	brief General function used to convert a string into a boolean equivalent value. param[in] str string input for conversion.
Method	Termination (see page 293)	brief Runs upon application exit. It saves the idi_dataset (see page 302) data structure.
Method	Time_Current_String (see page 294)	brief Obtains the current time and date. This is exerpetted from AES Universal Library/Driver. Excerpt from AES advanced Linux library/driver. COPYRIGHT NOTICE Copyright (c) 2012 by Apex Embedded Systems. param[out] buf resulting date-time string. param[in] buf_size the character size of the buffer (see page 325) including null terminating character.

Legend

Method	Method
--------	--------

Macros

Name	Description
AS_ADS1259_REGISTER_QTY (see page 340)	brief Store parameters in this data structure and then can easily read/write them.
BANK_EXTRACT_DEFINITION (see page 340)	This is macro BANK_EXTRACT_DEFINITION.

BANK_EXTRACT_ENUM (see page 341)	
BANK_INFO_DEFINITION (see page 341)	This is macro BANK_INFO_DEFINITION.
BANK_NULL_DEFINITION (see page 341)	This is macro BANK_NULL_DEFINITION.
BUFFER_COLUMN (see page 342)	This is macro BUFFER_COLUMN.
BUFFER_LENGTH (see page 342)	This is macro BUFFER_LENGTH.
CLOCK_PERIOD_SEC (see page 342)	brief Board clock period as defined by the on-board oscillator which is 50MHz
CMD_MAIN_EXTRACT_BOARD_TYPE_REQUIRED (see page 342)	This is macro CMD_MAIN_EXTRACT_BOARD_TYPE_REQUIRED.
CMD_MAIN_EXTRACT_COMMANDS (see page 343)	This is macro CMD_MAIN_EXTRACT_COMMANDS.
CMD_MAIN_LIST (see page 343)	brief Used to build out the command list data structure and the board type requirement list. Implemented in this fashion in order to keep all the data together.
DIN_TEST_DEBUG_PRINT (see page 344)	brief
EC_EXTRACT_ENUM (see page 344)	
EC_EXTRACT_HUMAN_READABLE (see page 344)	This is macro EC_EXTRACT_HUMAN_READABLE.
EC_HUMAN_READABLE_TERMINATE (see page 345)	This is macro EC_HUMAN_READABLE_TERMINATE.
ERROR_CODES (see page 345)	brief An organized error code listing. This macro is used to build the complete error code enumeration list.
false (see page 346)	This is macro false.
FRAM_BLOCK_SIZE (see page 346)	brief
IAI_REGISTER_SET_DEFINITION (see page 346)	
IDI_BANK_INFO_DEFINITION (see page 348)	brief The bank register mapping. This mapping is upwardly compatible with the legacy hardware banking register. It also allows for future expansion utilizing the lower bits which are currently unused.
IDI_BOARD_INIT_FILE_NAME (see page 348)	brief Board type information
IDI_DIN_GROUP_QTY (see page 349)	This is macro IDI_DIN_GROUP_QTY.
IDI_DIN_GROUP_SIZE (see page 349)	
IDI_DIN_QTY (see page 349)	This is macro IDI_DIN_QTY.
IDI_DIN_SHIFT_RIGHT (see page 349)	This is macro IDI_DIN_SHIFT_RIGHT.
IDI_IO_DIRECTION_TEST (see page 350)	brief Produces an I/O direction error report.
IDI_REGISTER_SET_DEFINITION (see page 350)	brief Organized list of registers and associate attributes of each of the registers. This macro does not consume any memory of its own, and is only 'consumed' or used to automatically build enumerations and pre-built data structures.
IDO_BANK_INFO_DEFINITION (see page 352)	This is macro IDO_BANK_INFO_DEFINITION.
IDO_BOARD_INIT_FILE_NAME (see page 353)	This is macro IDO_BOARD_INIT_FILE_NAME.
IDO_DO_GROUP_QTY (see page 353)	This is macro IDO_DO_GROUP_QTY.
IDO_DO_GROUP_SIZE (see page 353)	
IDO_DO_QTY (see page 353)	This is macro IDO_DO_QTY.
IDO_DO_SHIFT_RIGHT (see page 354)	This is macro IDO_DO_SHIFT_RIGHT.
IDO_REGISTER_SET_DEFINITION (see page 354)	This is macro IDO_REGISTER_SET_DEFINITION.
INT32_MAX (see page 355)	This is macro INT32_MAX.
INT32_MIN (see page 355)	This is macro INT32_MIN.
INTERRUPT (see page 356)	This is macro INTERRUPT.
IO_TRACE_DEFINITION (see page 356)	Trace triggers

IO_TRACE_DUMP_FILE_NAME (see page 356)	This is macro IO_TRACE_DUMP_FILE_NAME.
IO_TRACE_EXTRACT_ENUM (see page 357)	This is macro IO_TRACE_EXTRACT_ENUM.
IO_TRACE_EXTRACT_NAME (see page 357)	This is macro IO_TRACE_EXTRACT_NAME.
IO_TRACE_FILE_NAME_SIZE (see page 357)	This is macro IO_TRACE_FILE_NAME_SIZE.
IO_TRACE_SIZE (see page 357)	
IO_TRACE_START_DEFINITION (see page 358)	brief Refer to IO_Trace_Log_Dataset (see page 212)() function for action details. We put all this in the x-macro as a convenient summary.
IO_TRACE_START_STOP_EXTRACT_ENUM (see page 358)	
IO_TRACE_START_STOP_EXTRACT_NAME (see page 358)	This is macro IO_TRACE_START_STOP_EXTRACT_NAME.
IO_TRACE_START_STOP_EXTRACT_NAME_HELP_LIST (see page 359)	
IO_TRACE_START_STOP_EXTRACT_SWITCH_STATEMENT (see page 359)	This is macro IO_TRACE_START_STOP_EXTRACT_SWITCH_STATEMENT.
IO_TRACE_STOP_DEFINITION (see page 359)	This is macro IO_TRACE_STOP_DEFINITION.
IO_TRACE_USE_AS_LOGGING (see page 360)	This is macro IO_TRACE_USE_AS_LOGGING.
IO_TRACE_USE_FRAM_LOGGING (see page 360)	This is macro IO_TRACE_USE_FRAM_LOGGING.
IO_TRACE_USE_SPI_LOGGING (see page 360)	
IOKERN_DOS_INT_GET (see page 360)	
IOKERN_DOS_INT_SET (see page 361)	This is macro IOKERN_DOS_INT_SET.
IOKERN_DOS_NSEOI (see page 361)	End Of Interrupt
IOKERN_DOS_PIC1_BASE_ADDRESS (see page 361)	These are the port addresses of the 8259 Programmable Interrupt Controller (PIC).
IOKERN_DOS_PIC1_CMD (see page 362)	PIC1 Command Port
IOKERN_DOS_PIC1_IMR (see page 362)	PIC1 interrupt mask port
IOKERN_DOS_PIC2_BASE_ADDRESS (see page 362)	This is macro IOKERN_DOS_PIC2_BASE_ADDRESS.
IOKERN_DOS_PIC2_CMD (see page 362)	PIC2 Command Port
IOKERN_DOS_PIC2_IMR (see page 363)	PIC2 interrupt mask port
IOKERN_IRQ_CHAIN_SELECT (see page 363)	This is macro IOKERN_IRQ_CHAIN_SELECT.
IOKERN_IRQ_ENABLE (see page 364)	This is macro IOKERN_IRQ_ENABLE.
IOKERN_IRQ_END (see page 364)	This is macro IOKERN_IRQ_END.
IOKERN_IRQ_START (see page 364)	This is macro IOKERN_IRQ_START.
IOKERN_LOCAL_IRQ_RESTORE (see page 364)	This is macro IOKERN_LOCAL_IRQ_RESTORE.
IOKERN_LOCAL_IRQ_SAVE (see page 365)	
IOKERN_TASK_QTY (see page 365)	This is macro IOKERN_TASK_QTY.
MESSAGE_SIZE (see page 366)	This is macro MESSAGE_SIZE.
REG_EXTRACT_DEFINITION (see page 366)	brief Use to build the register definitions of the IDI and IDO boards.
REG_EXTRACT_ENUM (see page 366)	brief Use to build the register enumerations of the IDI and IDO boards.
REG_LOCATION_CHANNEL_GET (see page 367)	This is macro REG_LOCATION_CHANNEL_GET.
REG_LOCATION_LOGICAL_GET (see page 367)	

REG_LOCATION_SET (see page 367)	brief 'Index' is the row number, in this case it is the 'logical address' column in the register definition x-macros. The 'channel' is an index for channel information which in this case we are setting to zero as it is not required since that information is already embodied within the logical address for the specific register definition.
REGS_INIT_QTY (see page 367)	This is macro REGS_INIT_QTY.
SPI_BLOCK_SIZE (see page 368)	This is macro SPI_BLOCK_SIZE.
SPI_COMMIT_BIT_POSITION (see page 368)	brief
SPI_REGISTER_SET_DEFINITION (see page 368)	This is macro SPI_REGISTER_SET_DEFINITION.
strcmpl (see page 371)	kbhitt() and getch()
SVN_REV (see page 372)	brief The Subversion (SVN) time/date marker which is updated during commit of the source file to the repository.
TARGET_CPU_INTEL_386 (see page 372)	UINT_MAX
true (see page 372)	This is macro true.
UINT32_MAX (see page 372)	This is macro UINT32_MAX.

Structures

	Name	Description
◆	as_ads1259_registers (see page 297)	This is record as_ads1259_registers.
◆	bank_info (see page 297)	This is record bank_info.
◆	bit_string_info (see page 297)	brief
◆	board_dataset (see page 298)	brief Provides a generic interface to all board datasets. This information is used by several functions in order to determine how to read/write registers, etc.
◆	board_definition (see page 298)	This is record board_definition.
◆	command_line (see page 300)	brief Data structure used to decode command line operation.
◆	command_line_board (see page 300)	This is record command_line_board.
◆	din_cfg (see page 301)	This is record din_cfg.
◆	dout_cfg (see page 301)	This is record dout_cfg.
◆	ec_human_readable (see page 301)	Error code to human readable data structure
◆	idi_bank_info (see page 302)	list of "bank" names and associated enumerated symbol
◆	idi_dataset (see page 302)	
◆	idi_reg_definition (see page 303)	a read only list of register parameters
◆	ido_bank_info (see page 304)	This is record ido_bank_info.
◆	ido_dataset (see page 304)	
◆	ido_reg_definition (see page 305)	
◆	io_trace (see page 306)	
◆	io_trace_info (see page 306)	
◆	IOKERN_TASK_TYPE (see page 307)	
◆	port_list (see page 309)	
◆	pt_regs (see page 309)	This is record pt_regs.
◆	reg_definition (see page 310)	< Generic definition used in IO_Write_U8 (see page 217) and IO_Read_U8 (see page 206)
◆	spi_cfg (see page 310)	
◆	spi_dataset (see page 311)	

	spi_status (see page 311)	
---	---------------------------	--

Legend

	Structure
---	-----------

Types

Name	Description
BOOL (see page 318)	brief Boolean logic definitions
int16_t (see page 318)	DOS only
int32_t (see page 319)	DOS only
int8_t (see page 319)	This is type int8_t.
IOKERN_DOS_VECTOR_TYPE (see page 319)	
IOKERN_HELP_FP (see page 319)	This is type IOKERN_HELP_FP.
IOKERN_ISR_FP (see page 320)	This is type IOKERN_ISR_FP.
IOKERN_TASK_FP (see page 320)	
irqreturn_t (see page 320)	irqreturn_t is directly from Linux kernel
uint16_t (see page 321)	DOS only
uint32_t (see page 321)	DOS only
uint8_t (see page 321)	brief The C89 compiler is typically void of these definitions, so we include them here. Defines specific data width information. The idea is to make this target independent.

Variables

Name	Description
ADS1259_CLOCK_MHZ (see page 323)	static BOOL (see page 318) as_checksum = false;
ADS1259_CLOCK_TOLERANCE (see page 323)	2
ADS1259_FSC_MAX (see page 324)	0x00800000 => x2
ADS1259_FSC_MIN (see page 324)	0x00000000 => x0
ADS1259_OFC_MAX (see page 324)	0x007FFFFF
ADS1259_OFC_MIN (see page 324)	0xFF800001
as_bit_string (see page 325)	brief
as_register_name (see page 325)	brief
buffer (see page 325)	This is variable buffer.
buffer_index (see page 326)	This is variable buffer_index.
cmd_as (see page 326)	brief
cmd_fram (see page 327)	brief
cmd_iai16 (see page 327)	brief
cmd_idi48 (see page 327)	brief
cmd_idi48_test (see page 328)	brief
cmd_ido48 (see page 328)	brief
cmd_ido48_test (see page 328)	brief
cmd_loop (see page 329)	brief
cmd_prefix (see page 329)	brief
cmd_set (see page 329)	brief

cmd_spi (see page 330)	brief
cmd_top (see page 330)	brief
cmd_top_board_type_required (see page 331)	This is variable cmd_top_board_type_required.
cmd_trace (see page 331)	brief
ec_unknown (see page 331)	brief Translates an error code into a human readable message. Excerpt from AES advanced Linux library/driver. COPYRIGHT NOTICE Copyright (c) 2012 by Apex Embedded Systems. param[in] error_code The error code to be translated into a human readable message
global_board_id (see page 332)	This is variable global_board_id.
global_io_trace_buf (see page 332)	
global_io_trace_enable (see page 332)	This is variable global_io_trace_enable.
global_io_trace_name (see page 332)	This is variable global_io_trace_name.
global_io_trace_start_name (see page 333)	This is variable global_io_trace_start_name.
global_io_trace_stop_name (see page 333)	This is variable global_io_trace_stop_name.
global_irq.Please_install_handler_request (see page 333)	This is variable global_irq.Please_install_handler_request.
global_loop_command (see page 334)	This is variable global_loop_command.
global_loop_count (see page 334)	This is variable global_loop_count.
global_loop_count_counter (see page 334)	This is variable global_loop_count_counter.
global_loop_delay_ms (see page 334)	This is variable global_loop_delay_ms.
global_loop_space (see page 335)	This is variable global_loop_space.
idi_ds (see page 335)	Global data structure which is restored during initialization and saved during application termination.
ido_ds (see page 335)	
iokern_isr_table (see page 336)	
iokern_task (see page 336)	This is variable iokern_task.
lpm_lx800_port_list (see page 336)	This is variable lpm_lx800_port_list.
svn_revision_string (see page 337)	Software revision as determined by subversion commits

3.2.6.2 stopwatch.h

TO DO

COPYRIGHT Copyright(c) 1999-2008 by Apex Embedded Systems. All rights reserved. This document is the confidential property of Apex Embedded Systems Any reproduction or dissemination is prohibited unless specifically authorized in writing.

Body Source

```

1:
16: #ifndef __STOPWATCH_H__
17: #define __STOPWATCH_H__
18:
19: #ifdef __cplusplus
20: extern "C" {
21: #endif
22:
23:
24: #include <stdio.h>

```

```

25: #include <stdlib.h>
26: #include <string.h>
27:
28:
29: #include <time.h>
30:
31: #if ( defined( __GNUC__ ) )
32: #include <sys/time.h>
33: #define STOPWATCH_HANDLE_TYPE int
34: #define STOPWATCH_STATE_TYPE int
35: #elif ( defined(__BORLANDC__) || defined( _MSC_VER ) || defined(WIN32_DLL) || defined(
36: _DLL ) )
36: struct timeval {
37:     long    tv_sec;
38:     long    tv_usec;
39: };
40: #define STOPWATCH_HANDLE_TYPE char
41: #define STOPWATCH_STATE_TYPE char
42: #elif ( defined(_CC51) || defined(__C51__) || defined(__CX51__ ) )
43: #else
44: #endif
45:
46: typedef struct timeval STOPWATCH_TIMEVAL_TYPE;
47:
48:
49:
50: #define STOPWATCH_INACTIVE -1
51: #define STOPWATCH_ERROR -1
52: #define STOPWATCH_QUANTITY 4
53:
54: #define STOPWATCH_RESET 0
55: #define STOPWATCH_START 1
56: #define STOPWATCH_PAUSE 2
57:
58:
59: extern STOPWATCH_HANDLE_TYPE StopWatch_Open( void );
60: extern void StopWatch_Close( STOPWATCH_HANDLE_TYPE handle );
61: extern void StopWatch_Set( STOPWATCH_HANDLE_TYPE handle, STOPWATCH_STATE_TYPE state );
62: extern STOPWATCH_TIMEVAL_TYPE * StopWatch_Value( STOPWATCH_HANDLE_TYPE handle );
63: extern void StopWatch_TimeStamp_String( char * return_string, int return_string_size );
64: extern void StopWatch_Process( void );
65: extern void StopWatch_Termination( void );
66: extern void StopWatch_Initialization( void );
67:
68: #ifdef __cplusplus
69: }
70: #endif
71:
72: #endif

```

Functions

	Name	Description
💡	StopWatch_Close ([see page 290])	This is function StopWatch_Close.
💡	StopWatch_Initialization ([see page 291])	This is function StopWatch_Initialization.
💡	StopWatch_Open ([see page 291])	This is function StopWatch_Open.
💡	StopWatch_Process ([see page 291])	This is function StopWatch_Process.
💡	StopWatch_Set ([see page 291])	This is function StopWatch_Set.
💡	StopWatch_Termination ([see page 292])	This is function StopWatch_Termination.
💡	StopWatch_TimeStamp_String ([see page 292])	This is function StopWatch_TimeStamp_String.

	StopWatch_Value (see page 292)	This is function StopWatch_Value.
---	--------------------------------	-----------------------------------

Legend

	Method
---	--------

Macros

Name	Description
__STOPWATCH_H__ (see page 340)	This is macro __STOPWATCH_H__.
STOPWATCH_ERROR (see page 369)	This is macro STOPWATCH_ERROR.
STOPWATCH_HANDLE_TYPE (see page 369)	This is macro STOPWATCH_HANDLE_TYPE.
STOPWATCH_INACTIVE (see page 370)	This is macro STOPWATCH_INACTIVE.
STOPWATCH_PAUSE (see page 370)	This is macro STOPWATCH_PAUSE.
STOPWATCH_QUANTITY (see page 370)	This is macro STOPWATCH_QUANTITY.
STOPWATCH_RESET (see page 370)	This is macro STOPWATCH_RESET.
STOPWATCH_START (see page 371)	This is macro STOPWATCH_START.
STOPWATCH_STATE_TYPE (see page 371)	This is macro STOPWATCH_STATE_TYPE.

Structures

	Name	Description
	timeval (see page 312)	This is record timeval.

Legend

	Structure
---	-----------

Types

Name	Description
STOPWATCH_TIMEVAL_TYPE (see page 321)	This is type STOPWATCH_TIMEVAL_TYPE.

Index

_STOPWATCH_H_ 340

_STOPWATCH_H_ macro 340

A

ADS1259_Calibration_Range_Test 28

ADS1259_Calibration_Range_Test function 28

ADS1259_CLOCK_MHZ 323

ADS1259_CLOCK_MHZ variable 323

ADS1259_CLOCK_TOLERANCE 323

ADS1259_CLOCK_TOLERANCE variable 323

ADS1259_FSC_MAX 324

ADS1259_FSC_MAX variable 324

ADS1259_FSC_MIN 324

ADS1259_FSC_MIN variable 324

ADS1259_OFC_MAX 324

ADS1259_OFC_MAX variable 324

ADS1259_OFC_MIN 324

ADS1259_OFC_MIN variable 324

AS_ADS1259_REGISTER_QTY 340

AS_ADS1259_REGISTER_QTY macro 340

as_ads1259_registers 297

as_ads1259_registers structure 297

as_bit_string 325

as_bit_string variable 325

AS_Calibration_Value_Read 29

AS_Calibration_Value_Read function 29

AS_Calibration_Value_Write 30

AS_Calibration_Value_Write function 30

AS_Chip_Select_Route_Override 31

AS_Chip_Select_Route_Override function 31

AS_Chip_Select_Route_Restore 31

AS_Chip_Select_Route_Restore function 31

AS_Data_Read 32

AS_Data_Read function 32

AS_Data_Ready_NoWait 33

AS_Data_Ready_NoWait function 33

AS_Data_Ready_Wait 33

AS_Data_Ready_Wait function 33

AS_Opcode_Write 34

AS_Opcode_Write function 34

AS_Rdata_Statistics 35

AS_Rdata_Statistics function 35

AS_Register_Defaults 36

AS_Register_Defaults function 36

as_register_name 325

as_register_name variable 325

AS_Register_Name_To_Offset 36

AS_Register_Name_To_Offset function 36

AS_Registers_Read 37

AS_Registers_Read function 37

AS_Registers_Write 38

AS_Registers_Write function 38

AS_Time_Statistics 39

AS_Time_Statistics function 39

B

BANK_ENUM 312

BANK_ENUM enumeration 312

BANK_EXTRACT_DEFINITION 340

BANK_EXTRACT_DEFINITION macro 340

BANK_EXTRACT_ENUM 341

BANK_EXTRACT_ENUM macro 341

bank_info 297

bank_info structure 297

BANK_INFO_DEFINITION 341

BANK_INFO_DEFINITION macro 341

Bank_Name_To_Symbol 40

Bank_Name_To_Symbol function 40

BANK_NULL_DEFINITION 341

BANK_NULL_DEFINITION macro 341

Bank_Symbol_To_Name 41

Bank_Symbol_To_Name function 41

Bit_String_Index_By_Name 41
Bit_String_Index_By_Name function 41
bit_string_info 297
bit_string_info structure 297
Bit_String_Report 42
Bit_String_Report function 42
board_dataset 298
board_dataset structure 298
board_definition 298
board_definition structure 298
BOOL 318
BOOL type 318
buffer 325
buffer variable 325
buffer_column 299
BUFFER_COLUMN 342
buffer_column enumeration 299
BUFFER_COLUMN macro 342
BUFFER_COLUMN_TAG enumeration member 299
BUFFER_COLUMN_VALUE enumeration member 299
buffer_index 326
buffer_index variable 326
Buffer_Init 43
Buffer_Init function 43
Buffer_Length 43
BUFFER_LENGTH 342
Buffer_Length function 43
BUFFER_LENGTH macro 342
Buffer_Maximum_Int32 44
Buffer_Maximum_Int32 function 44
Buffer_Mean_Int32 44
Buffer_Mean_Int32 function 44
Buffer_Minimum_Int32 45
Buffer_Minimum_Int32 function 45
Buffer_Peak_To_Peak_Int32 45
Buffer_Peak_To_Peak_Int32 function 45
Buffer_Save 46
Buffer_Save function 46
Buffer_Save_Binary_Int32 46
Buffer_Save_Binary_Int32 function 46
Buffer_Standard_Deviation_Int32 47
Buffer_Standard_Deviation_Int32 function 47
Buffer_Stuff 48
Buffer_Stuff function 48

C

C Test Code - DOS 5
Character_Get 48
Character_Get function 48
CLM_BTS 299
CLM_BTS enumeration 299
CLM_BTS_IMPLIED enumeration member 299
CLM_BTS_NORMAL enumeration member 299
CLM_BTS_NOT_REQUIRED enumeration member 299
CLOCK_PERIOD_SEC 342
CLOCK_PERIOD_SEC macro 342
CMD_AS_Calibration_Gain 49
CMD_AS_Calibration_Gain function 49
CMD_AS_Calibration_Offset 50
CMD_AS_Calibration_Offset function 50
CMD_AS_Gancal 51
CMD_AS_Gancal function 51
CMD_AS_Ofscal 51
CMD_AS_Ofscal function 51
CMD_AS_Parameter 52
CMD_AS_Parameter function 52
CMD_AS_Rdata 54
CMD_AS_Rdata function 54
CMD_AS_Rdatac 58
CMD_AS_Rdatac function 58
CMD_AS_Reg_Write_By_Name 58
CMD_AS_Reg_Write_By_Name function 58
CMD_AS_Register_Load 60
CMD_AS_Register_Load function 60
CMD_AS_Register_Save 61
CMD_AS_Register_Save function 61

CMD__AS_Reset	61	CMD__IAI16_AI_Channel	75
CMD__AS_Reset function	61	CMD__IAI16_AI_Channel function	75
CMD__AS_RReg	62	CMD__IAI16_AI_ID	75
CMD__AS_RReg function	62	CMD__IAI16_AI_ID function	75
CMD__AS_Sdatac	64	CMD__IDI48_DIN_All	76
CMD__AS_Sdatac function	64	CMD__IDI48_DIN_All function	76
CMD__AS_Sleep	64	CMD__IDI48_DIN_Channel	77
CMD__AS_Sleep function	64	CMD__IDI48_DIN_Channel function	77
CMD__AS_Start	65	CMD__IDI48_DIN_Group	78
CMD__AS_Start function	65	CMD__IDI48_DIN_Group function	78
CMD__AS_Stop	65	CMD__IDI48_DIN_Helper_All_Values_False	79
CMD__AS_Stop function	65	CMD__IDI48_DIN_Helper_All_Values_False function	79
CMD__AS_Wakeup	66	CMD__IDI48_DIN_ID	79
CMD__AS_Wakeup function	66	CMD__IDI48_DIN_ID function	79
CMD__AS_WReg	66	CMD__IDI48_DIN_Test	80
CMD__AS_WReg function	66	CMD__IDI48_DIN_Test function	80
CMD__FRAM_Dump	67	CMD__IDI48_DIN_Test_FE	81
CMD__FRAM_Dump function	67	CMD__IDI48_DIN_Test_FE function	81
CMD__FRAM_Init	68	CMD__IDI48_DIN_Test_Interrupt	84
CMD__FRAM_Init function	68	CMD__IDI48_DIN_Test_Interrupt function	84
CMD__FRAM_Load	69	CMD__IDI48_DIN_Test_Interrupt_Handler	89
CMD__FRAM_Load function	69	CMD__IDI48_DIN_Test_Interrupt_Handler function	89
CMD__FRAM_RDID	70	CMD__IDI48_DIN_Test_RE	90
CMD__FRAM_RDID function	70	CMD__IDI48_DIN_Test_RE function	90
CMD__FRAM_RDSR	70	CMD__IDI48_DIN_Test_Value	93
CMD__FRAM_RDSR function	70	CMD__IDI48_DIN_Test_Value function	93
CMD__FRAM_Save	71	CMD__IDO48_DO_All	96
CMD__FRAM_Save function	71	CMD__IDO48_DO_All function	96
CMD__FRAM_WRDI	72	CMD__IDO48_DO_Channel	97
CMD__FRAM_WRDI function	72	CMD__IDO48_DO_Channel function	97
CMD__FRAM_WREN	72	CMD__IDO48_DO_Group	98
CMD__FRAM_WREN function	72	CMD__IDO48_DO_Group function	98
CMD__FRAM_Write	73	CMD__IDO48_DO_ID	99
CMD__FRAM_Write function	73	CMD__IDO48_DO_ID function	99
CMD__FRAM_WRSR	73	CMD__IDO48_DOUT_Test_Alternate	100
CMD__FRAM_WRSR function	73	CMD__IDO48_DOUT_Test_Alternate function	100
CMD__IAI16_AI_All	74	CMD__IDO48_DOUT_Test_One_Hot	102
CMD__IAI16_AI_All function	74	CMD__IDO48_DOUT_Test_One_Hot function	102

CMD__IDO48_IDI48_Loopback	103	CMD__SPI_Config_SDIO_Wrap	125
CMD__IDO48_IDI48_Loopback function	103	CMD__SPI_Config_SDIO_Wrap function	125
CMD__IDO48_Test	109	CMD__SPI_Config_SDO_Polarity	126
CMD__IDO48_Test function	109	CMD__SPI_Config_SDO_Polarity function	126
CMD__Main_AnalogStick_CS	110	CMD__SPI_Data	126
CMD__Main_AnalogStick_CS function	110	CMD__SPI_Data function	126
CMD__Main_Base	110	CMD__SPI_Data_Interpreter	128
CMD__Main_Base function	110	CMD__SPI_Data_Interpreter function	128
CMD__Main_FRAM_CS	111	CMD__SPI_FIFO	129
CMD__Main_FRAM_CS function	111	CMD__SPI_FIFO function	129
CMD__Main_I_Count	112	CMD__SPI_ID	132
CMD__Main_I_Count function	112	CMD__SPI_ID function	132
CMD__Main_Init_Reg	113	CMD__SPI_Status	132
CMD__Main_Init_Reg function	113	CMD__SPI_Status function	132
CMD__Main_IO_Behavior	114	CMD__Trace_File	133
CMD__Main_IO_Behavior function	114	CMD__Trace_File function	133
CMD__Main_Irq_Number	115	CMD__Trace_Start	134
CMD__Main_Irq_Number function	115	CMD__Trace_Start function	134
CMD__Main_Mode_Jumpers	116	CMD__Trace_Stop	135
CMD__Main_Mode_Jumpers function	116	CMD__Trace_Stop function	135
CMD__Main_Trace	117	cmd_as	326
CMD__Main_Trace function	117	cmd_as variable	326
CMD__SPI_Commit	118	cmd_fram	327
CMD__SPI_Commit function	118	cmd_fram variable	327
CMD__SPI_Config_Chip_Select_Behavior	118	cmd_iai16	327
CMD__SPI_Config_Chip_Select_Behavior function	118	cmd_iai16 variable	327
CMD__SPI_Config_Chip_Select_Route	120	cmd_idi48	327
CMD__SPI_Config_Chip_Select_Route function	120	cmd_idi48 variable	327
CMD__SPI_Config_Clock_Hz	120	cmd_idi48_test	328
CMD__SPI_Config_Clock_Hz function	120	cmd_idi48_test variable	328
CMD__SPI_Config_End_Cycle_Delay_Sec	121	cmd_ido48	328
CMD__SPI_Config_End_Cycle_Delay_Sec function	121	cmd_ido48 variable	328
CMD__SPI_Config_Get	122	cmd_ido48_test	328
CMD__SPI_Config_Get function	122	cmd_ido48_test variable	328
CMD__SPI_Config_Mode	123	cmd_loop	329
CMD__SPI_Config_Mode function	123	cmd_loop variable	329
CMD__SPI_Config_SDI_Polarity	124	CMD_MAIN_EXTRACT_BOARD_TYPE_REQUIRED	342
CMD__SPI_Config_SDI_Polarity function	124	CMD_MAIN_EXTRACT_BOARD_TYPE_REQUIRED macro	

342	Command_Line_IO_Read function 143
CMD_MAIN_EXTRACT_COMMANDS 343	Command_Line_IO_Write 145
CMD_MAIN_EXTRACT_COMMANDS macro 343	Command_Line_IO_Write function 145
CMD_MAIN_LIST 343	Command_Line_Loop_Count 146
CMD_MAIN_LIST macro 343	Command_Line_Loop_Count function 146
cmd_prefix 329	Command_Line_Loop_Delay 147
cmd_prefix variable 329	Command_Line_Loop_Delay function 147
cmd_set 329	Command_Line_Loop_Space 148
cmd_set variable 329	Command_Line_Loop_Space function 148
cmd_spi 330	Command_Line_Main 148
cmd_spi variable 330	Command_Line_Main function 148
cmd_top 330	Command_Line_Main__Board_Type_Status 149
cmd_top variable 330	Command_Line_Main__Board_Type_Status function 149
cmd_top__board_type_required 331	Command_Line_Prefix 150
cmd_top__board_type_required variable 331	Command_Line_Prefix function 150
cmd_trace 331	Command_Line_Prefix_Irq 151
cmd_trace variable 331	Command_Line_Prefix_Irq function 151
command_line 300	Command_Line_Prefix_Loop 152
command_line structure 300	Command_Line_Prefix_Loop function 152
Command_Line_Analog_Stick 136	Command_Line_Prefix_Trace 153
Command_Line_Analog_Stick function 136	Command_Line_Prefix_Trace function 153
command_line_board 300	Command_Line_Register_Transaction 154
command_line_board structure 300	Command_Line_Register_Transaction function 154
Command_Line_Dump 137	Command_Line_Set 156
Command_Line_Dump function 137	Command_Line_Set function 156
Command_Line_FPGA 138	Command_Line_SPI 157
Command_Line_FPGA function 138	Command_Line_SPI function 157
Command_Line_FRAM 139	Command_Line_Wait 158
Command_Line_FRAM function 139	Command_Line_Wait function 158
Command_Line_Help 140	Commands 5
Command_Line_Help function 140	
Command_Line_IAI16 140	
Command_Line_IAI16 function 140	
Command_Line_IDI48 141	D
Command_Line_IDI48 function 141	
Command_Line_IDO48 142	din_cfg 301
Command_Line_IDO48 function 142	din_cfg structure 301
Command_Line_IO_Read 143	DIN_TEST_DEBUG_PRINT 344
	DIN_TEST_DEBUG_PRINT macro 344
	dout_cfg 301
	dout_cfg structure 301

E

EC_Code_To_Human_Readable 158
EC_Code_To_Human_Readable function 158
EC_ENUM 313
EC_ENUM enumeration 313
EC_EXTRACT_ENUM 344
EC_EXTRACT_ENUM macro 344
EC_EXTRACT_HUMAN_READABLE 344
EC_EXTRACT_HUMAN_READABLE macro 344
ec_human_readable 301
ec_human_readable structure 301
EC_HUMAN_READABLE_TERMINATE 345
EC_HUMAN_READABLE_TERMINATE macro 345
ec_unknown 331
ec_unknown variable 331
ERROR_CODES 345
ERROR_CODES macro 345

F

false 346
false macro 346
Files 373
FPGA_Date_Time_Information 159
FPGA_Date_Time_Information function 159
FRAM_Chip_Select_Route_Override 161
FRAM_Chip_Select_Route_Override function 161
FRAM_Chip_Select_Route_Restore 162
FRAM_Chip_Select_Route_Restore function 162
FRAM_Memory_Read 162
FRAM_Memory_Read function 162
FRAM_Memory_Write 164
FRAM_Memory_Write function 164
FRAM_Read_ID 165
FRAM_Read_ID function 165
FRAM_Read_Status_Register 166
FRAM_Read_Status_Register function 166

FRAM_Write_Disable 167
FRAM_Write_Disable function 167
FRAM_Write_Enable_Latch_Set 168
FRAM_Write_Enable_Latch_Set function 168
FRAM_Write_Status_Register 169
FRAM_Write_Status_Register function 169
FRAM_BLOCK_SIZE 346
FRAM_BLOCK_SIZE macro 346
FRAM_File_To_Memory 170
FRAM_File_To_Memory function 170
FRAM_Memory_To_File 171
FRAM_Memory_To_File function 171
FRAM_Report 172
FRAM_Report function 172
FRAM_Set 173
FRAM_Set function 173
Functions 15

G

global_board_id 332
global_board_id variable 332
global_io_trace_buf 332
global_io_trace_buf variable 332
global_io_trace_enable 332
global_io_trace_enable variable 332
global_io_trace_name 332
global_io_trace_name variable 332
global_io_trace_start_name 333
global_io_trace_start_name variable 333
global_io_trace_stop_name 333
global_io_trace_stop_name variable 333
global_irq_please_install_handler_request 333
global_irq_please_install_handler_request variable 333
global_loop_command 334
global_loop_command variable 334
global_loop_count 334
global_loop_count variable 334
global_loop_count_counter 334

global_loop_count_counter variable 334	idi_dataset 302
global_loop_delay_ms 334	idi_dataset structure 302
global_loop_delay_ms variable 334	IDI_Dataset_Defaults 182
global_loop_space 335	IDI_Dataset_Defaults function 182
global_loop_space variable 335	IDI_DIN_Channel_Get 183
	IDI_DIN_Channel_Get function 183
	IDI_DIN_Group_Get 183
	IDI_DIN_Group_Get function 183
H	IDI_DIN_GROUP_QTY 349
Help 175	IDI_DIN_GROUP_SIZE 349
Help function 175	IDI_DIN_GROUP_SIZE macro 349
Help_Output 176	IDI_DIN_ID_Get 184
Help_Output function 176	IDI_DIN_ID_Get function 184
Help_Pause_Helper 177	IDI_DIN_IsNotPresent 185
Help_Pause_Helper function 177	IDI_DIN_IsNotPresent function 185
Hex_Dump_Line 178	IDI_DIN_Pending_Clear 185
Hex_Dump_Line function 178	IDI_DIN_Pending_Clear function 185
	IDI_DIN_QTY 349
I	IDI_DIN_QTY macro 349
IAI_AI_ID_Get 179	IDI_DIN_SHIFT_RIGHT 349
IAI_AI_ID_Get function 179	IDI_DIN_SHIFT_RIGHT macro 349
IAI_REGISTER_SET_DEFINITION 346	idi_ds 335
IAI_REGISTER_SET_DEFINITION macro 346	idi_ds variable 335
IAI16_AI_Channel_Get 179	IDI_Help 186
IAI16_AI_Channel_Get function 179	IDI_Help function 186
IAI16_AI_Channel_Multiple_Get 180	IDI_Initialization_Data_Structure_Load 186
IAI16_AI_Channel_Multiple_Get function 180	IDI_Initialization_Data_Structure_Load function 186
idi.c 373	IDI_Initialization_Register 187
IDI_BANK_ENUM 313	IDI_Initialization_Register function 187
IDI_BANK_ENUM enumeration 313	IDI_IO_DIRECTION_TEST 350
idi_bank_info 302	IDI_IO_DIRECTION_TEST macro 350
idi_bank_info structure 302	IDI_Main_Loop_Testing 188
IDI_BANK_INFO_DEFINITION 348	IDI_Main_Loop_Testing function 188
IDI_BANK_INFO_DEFINITION macro 348	idi_reg_definition 303
IDI_Bank_Name_To_Symbol 181	idi_reg_definition structure 303
IDI_Bank_Name_To_Symbol function 181	IDI_REG_ENUM 313
IDI_Bank_Symbol_To_Name 181	IDI_REG_ENUM enumeration 313
IDI_Bank_Symbol_To_Name function 181	
IDI_BOARD_INIT_FILE_NAME 348	
IDI_BOARD_INIT_FILE_NAME macro 348	

IDI_Register_Report_CSV	189	IDO_DO_SHIFT_RIGHT	354
IDI_Register_Report_CSV	function 189	IDO_DO_SHIFT_RIGHT	macro 354
IDI_REGISTER_SET_DEFINITION	350	IDO_DOUT_ID_Get	196
IDI_REGISTER_SET_DEFINITION	macro 350	IDO_DOUT_ID_Get	function 196
IDI_Termination	190	IDO_DOUT_IsNotPresent	197
IDI_Termination	function 190	IDO_DOUT_IsNotPresent	function 197
IDI48_DIN_ID_Port_Scan	191	ido_ds	335
IDI48_DIN_ID_Port_Scan	function 191	ido_ds	variable 335
IDO_BANK_ENUM	314	IDO_Help	198
IDO_BANK_ENUM	enumeration 314	IDO_Help	function 198
ido_bank_info	304	IDO_Initialization_Data_Structure_Load	198
ido_bank_info	structure 304	IDO_Initialization_Data_Structure_Load	function 198
IDO_BANK_INFO_DEFINITION	352	IDO_Initialization_Register	199
IDO_BANK_INFO_DEFINITION	macro 352	IDO_Initialization_Register	function 199
IDO_Bank_Name_To_Symbol	192	ido_reg_definition	305
IDO_Bank_Name_To_Symbol	function 192	ido_reg_definition	structure 305
IDO_Bank_Symbol_To_Name	193	IDO_REG_ENUM	314
IDO_Bank_Symbol_To_Name	function 193	IDO_REG_ENUM	enumeration 314
IDO_BOARD_INIT_FILE_NAME	353	IDO_Register_Report_CSV	200
IDO_BOARD_INIT_FILE_NAME	macro 353	IDO_Register_Report_CSV	function 200
ido_dataset	304	IDO_REGISTER_SET_DEFINITION	354
ido_dataset	structure 304	IDO_REGISTER_SET_DEFINITION	macro 354
IDO_Dataset_Defaults	193	IDO_Termination	201
IDO_Dataset_Defaults	function 193	IDO_Termination	function 201
IDO_DO_Channel_Get	194	IDO48_IDI48_Loopback_Test	202
IDO_DO_Channel_Get	function 194	IDO48_IDI48_Loopback_Test	function 202
IDO_DO_Channel_Set	195	Initialization	202
IDO_DO_Channel_Set	function 195	Initialization	function 202
IDO_DO_Group_Get	195	int16_t	318
IDO_DO_Group_Get	function 195	int16_t	type 318
IDO_DO_GROUP_QTY	353	INT32_MAX	355
IDO_DO_GROUP_QTY	macro 353	INT32_MAX	macro 355
IDO_DO_Group_Set	196	INT32_MIN	355
IDO_DO_Group_Set	function 196	INT32_MIN	macro 355
IDO_DO_GROUP_SIZE	353	int32_t	319
IDO_DO_GROUP_SIZE	macro 353	int32_t	type 319
IDO_DO_QTY	353	int8_t	319
IDO_DO_QTY	macro 353	int8_t	type 319

INTERRUPT 356
INTERRUPT macro 356
IO_Direction_IsNotValid 204
IO_Direction_IsNotValid function 204
IO_Get_Symbol_Name 204
IO_Get_Symbol_Name function 204
IO_Read_U16_Address_Fixed 205
IO_Read_U16_Address_Fixed function 205
IO_Read_U16_Address_Increment 206
IO_Read_U16_Address_Increment function 206
IO_Read_U8 206
IO_Read_U8 function 206
IO_Read_U8_Port 208
IO_Read_U8_Port function 208
io_trace 306
io_trace structure 306
IO_TRACE_DEFINITION 356
IO_TRACE_DEFINITION macro 356
IO_Trace_Dump 208
IO_Trace_Dump function 208
IO_TRACE_DUMP_FILE_NAME 356
IO_TRACE_DUMP_FILE_NAME macro 356
IO_TRACE_ENUM 306
IO_TRACE_ENUM enumeration 306
IO_TRACE_EXTRACT_ENUM 357
IO_TRACE_EXTRACT_ENUM macro 357
IO_TRACE_EXTRACT_NAME 357
IO_TRACE_EXTRACT_NAME macro 357
IO_TRACE_FILE_NAME_SIZE 357
IO_TRACE_FILE_NAME_SIZE macro 357
io_trace_info 306
io_trace_info structure 306
IO_Trace_Initialize 211
IO_Trace_Initialize function 211
IO_Trace_Log 212
IO_Trace_Log function 212
IO_Trace_Log_Dataset 212
IO_Trace_Log_Dataset function 212
IO_Trace_Log_Dataset_Begin 213
IO_Trace_Log_Dataset_Begin function 213
IO_Trace_Log_Dataset_End 214
IO_Trace_Log_Dataset_End function 214
IO_Trace_Log_Function 215
IO_Trace_Log_Function function 215
IO_TRACE_SIZE 357
IO_TRACE_SIZE macro 357
IO_TRACE_START_DEFINITION 358
IO_TRACE_START_DEFINITION macro 358
IO_TRACE_START_ENUM 307
IO_TRACE_START_ENUM enumeration 307
IO_TRACE_START_STOP_EXTRACT_ENUM 358
IO_TRACE_START_STOP_EXTRACT_ENUM macro 358
IO_TRACE_START_STOP_EXTRACT_NAME 358
IO_TRACE_START_STOP_EXTRACT_NAME macro 358
IO_TRACE_START_STOP_EXTRACT_NAME_HELP_LIST 359
IO_TRACE_START_STOP_EXTRACT_NAME_HELP_LIST macro 359
IO_TRACE_START_STOP_EXTRACT_SWITCH_STATEMENT 359
IO_TRACE_START_STOP_EXTRACT_SWITCH_STATEMENT macro 359
IO_TRACE_STOP_DEFINITION 359
IO_TRACE_STOP_DEFINITION macro 359
IO_TRACE_STOP_ENUM 307
IO_TRACE_STOP_ENUM enumeration 307
IO_TRACE_USE_AS_LOGGING 360
IO_TRACE_USE_AS_LOGGING macro 360
IO_TRACE_USE_FRAM_LOGGING 360
IO_TRACE_USE_FRAM_LOGGING macro 360
IO_TRACE_USE_SPI_LOGGING 360
IO_TRACE_USE_SPI_LOGGING macro 360
IO_Write_U16_Address_Fixed 216
IO_Write_U16_Address_Fixed function 216
IO_Write_U16_Address_Increment 216
IO_Write_U16_Address_Increment function 216
IO_Write_U8 217

IO_Write_U8 function 217	IOKern_DOS_Vector_Set function 225
IO_Write_U8_Port 218	IOKERN_DOS_VECTOR_TYPE 319
IO_Write_U8_Port function 218	IOKERN_DOS_VECTOR_TYPE type 319
IOKERN_CHAIN_TO_OLD_ENUM 314	IOKERN_HELP_FP 319
IOKERN_CHAIN_TO_OLD_ENUM enumeration 314	IOKERN_HELP_FP type 319
IOKERN_DOS_INT_GET 360	IOKern_Interrupt_Helper 225
IOKERN_DOS_INT_GET macro 360	IOKern_Interrupt_Helper function 225
IOKERN_DOS_INT_SET 361	IOKERN_IRQ_CHAIN_SELECT 363
IOKERN_DOS_INT_SET macro 361	IOKERN_IRQ_CHAIN_SELECT macro 363
IOKern_DOS_IRQ_Free 219	IOKERN_IRQ_ENABLE 364
IOKern_DOS_IRQ_Free function 219	IOKERN_IRQ_ENABLE macro 364
IOKern_DOS_IRQ_Mask_Get 220	IOKERN_IRQ_END 364
IOKern_DOS_IRQ_Mask_Get function 220	IOKERN_IRQ_END macro 364
IOKern_DOS_IRQ_Mask_Set 220	IOKERN_IRQ_ENUM 315
IOKern_DOS_IRQ_Mask_Set function 220	IOKERN_IRQ_ENUM enumeration 315
IOKern_DOS_IRQ_Request 221	IOKern_IRQ_Invoke_ISR_Test 226
IOKern_DOS_IRQ_Request function 221	IOKern_IRQ_Invoke_ISR_Test function 226
IOKern_DOS_ISR_Install 222	IOKERN_IRQ_START 364
IOKern_DOS_ISR_Install function 222	IOKERN_IRQ_START macro 364
IOKern_DOS_ISR_Restore 223	IOKern_IRQ_Test 226
IOKern_DOS_ISR_Restore function 223	IOKern_IRQ_Test function 226
IOKERN_DOS_NSEOI 361	IOKern_IRQ_Test_Helper 227
IOKERN_DOS_NSEOI macro 361	IOKern_IRQ_Test_Helper function 227
IOKERN_DOS_PIC1_BASE_ADDRESS 361	IOKERN_ISR_FP 320
IOKERN_DOS_PIC1_BASE_ADDRESS macro 361	IOKERN_ISR_FP type 320
IOKERN_DOS_PIC1_CMD 362	ikern_isr_table 336
IOKERN_DOS_PIC1_CMD macro 362	ikern_isr_table variable 336
IOKERN_DOS_PIC1_IMR 362	IOKern_ISR0 228
IOKERN_DOS_PIC1_IMR macro 362	IOKern_ISR0 function 228
IOKERN_DOS_PIC2_BASE_ADDRESS 362	IOKern_ISR1 228
IOKERN_DOS_PIC2_BASE_ADDRESS macro 362	IOKern_ISR1 function 228
IOKERN_DOS_PIC2_CMD 362	IOKern_ISR10 229
IOKERN_DOS_PIC2_CMD macro 362	IOKern_ISR10 function 229
IOKERN_DOS_PIC2_IMR 363	IOKern_ISR11 229
IOKERN_DOS_PIC2_IMR macro 363	IOKern_ISR11 function 229
IOKern_DOS_Vector_Get 224	IOKern_ISR12 230
IOKern_DOS_Vector_Get function 224	IOKern_ISR12 function 230
IOKern_DOS_Vector_Set 225	IOKern_ISR13 230

IOKern_ISR13 function 230
 IOKern_ISR14 231
 IOKern_ISR14 function 231
 IOKern_ISR15 231
 IOKern_ISR15 function 231
 IOKern_ISR2 232
 IOKern_ISR2 function 232
 IOKern_ISR3 232
 IOKern_ISR3 function 232
 IOKern_ISR4 233
 IOKern_ISR4 function 233
 IOKern_ISR5 233
 IOKern_ISR5 function 233
 IOKern_ISR6 234
 IOKern_ISR6 function 234
 IOKern_ISR7 234
 IOKern_ISR7 function 234
 IOKern_ISR8 235
 IOKern_ISR8 function 235
 IOKern_ISR9 235
 IOKern_ISR9 function 235
 IOKERN_LOCAL_IRQ_RESTORE 364
 IOKERN_LOCAL_IRQ_RESTORE macro 364
 IOKERN_LOCAL_IRQ_SAVE 365
 IOKERN_LOCAL_IRQ_SAVE macro 365
 IOKern_PIC_EOI 236
 IOKern_PIC_EOI function 236
 IOKern_Resource_Initialization 236
 IOKern_Resource_Initialization function 236
 IOKern_Resource_Termination 237
 IOKern_Resource_Termination function 237
 iokern_task 336
 iokern_task variable 336
 IOKERN_TASK_FP 320
 IOKERN_TASK_FP type 320
 IOKern_Task_Handle_Get 238
 IOKern_Task_Handle_Get function 238
 IOKERN_TASK_ID_ENUM 316

IOKERN_TASK_ID_ENUM enumeration 316
 IOKern_Task_ID_Get 238
 IOKern_Task_ID_Get function 238
 IOKERN_TASK_QTY 365
 IOKERN_TASK_QTY macro 365
 IOKERN_TASK_TYPE 307
 IOKERN_TASK_TYPE structure 307
 IRQ_HANDLED enumeration member 308
 IRQ_NONE enumeration member 308
 IRQ_WAKE_THREAD enumeration member 308
 irqreturn 308
 irqreturn enumeration 308
 irqreturn_t 320
 irqreturn_t type 320

L

Legal Notice 3
 lpm_lx800_port_list 336
 lpm_lx800_port_list variable 336

M

Macros 337
 main 239
 main function 239
 Main_Versalogs 243
 Main_Versalogs function 243
 MESSAGE_SIZE 366
 MESSAGE_SIZE macro 366
 MODE_JUMPERS_ENUM 316
 MODE_JUMPERS_ENUM enumeration 316

P

port_list 309
 port_list structure 309
 Print_Byt_List 243
 Print_Byt_List function 243
 Print_Multiple 245

Print_Multiple function 245

pt_regs 309

pt_regs structure 309

R

reg_definition 310

reg_definition structure 310

REG_DIR_ENUM 317

REG_DIR_ENUM enumeration 317

REG_EXTRACT_DEFINITION 366

REG_EXTRACT_DEFINITION macro 366

REG_EXTRACT_ENUM 366

REG_EXTRACT_ENUM macro 366

REG_LOCATION_CHANNEL_GET 367

REG_LOCATION_CHANNEL_GET macro 367

REG_LOCATION_LOGICAL_GET 367

REG_LOCATION_LOGICAL_GET macro 367

REG_LOCATION_SET 367

REG_LOCATION_SET macro 367

Register_Acronym_To_Row 245

Register_Acronym_To_Row function 245

Register_Report_CSV 246

Register_Report_CSV function 246

REGS_INIT_QTY 367

REGS_INIT_QTY macro 367

S

Signal_Handler 247

Signal_Handler function 247

SPI_BLOCK_SIZE 368

SPI_BLOCK_SIZE macro 368

SPI_Calculate_Clock 247

SPI_Calculate_Clock function 247

SPI_Calculate_End_Cycle_Delay 248

SPI_Calculate_End_Cycle_Delay function 248

SPI_Calculate_Half_Clock 249

SPI_Calculate_Half_Clock function 249

SPI_Calculate_Half_Clock_Interval_Sec 251

SPI_Calculate_Half_Clock_Interval_Sec function 251

spi_cfg 310

spi_cfg structure 310

SPI_Commit 251

SPI_Commit function 251

SPI_COMMIT_BIT_POSITION 368

SPI_COMMIT_BIT_POSITION macro 368

SPI_Commit_Get 252

SPI_Commit_Get function 252

SPI_Commit_Is_Inactive 253

SPI_Commit_Is_Inactive function 253

SPI_Configuration_Chip_Select_Behavior_Get 253

SPI_Configuration_Chip_Select_Behavior_Get function 253

SPI_Configuration_Chip_Select_Behavior_Set 254

SPI_Configuration_Chip_Select_Behavior_Set function 254

SPI_Configuration_Chip_Select_Route_Get 255

SPI_Configuration_Chip_Select_Route_Get function 255

SPI_Configuration_Chip_Select_Route_Set 256

SPI_Configuration_Chip_Select_Route_Set function 256

SPI_Configuration_Get 257

SPI_Configuration_Get function 257

SPI_Configuration_Initialize 258

SPI_Configuration_Initialize function 258

SPI_Configuration_Set 259

SPI_Configuration_Set function 259

SPI_CS_B_ENUM 317

SPI_CS_B_ENUM enumeration 317

SPI_Data_Read 261

SPI_Data_Read function 261

SPI_Data_Write 262

SPI_Data_Write function 262

SPI_Data_Write_Read 263

SPI_Data_Write_Read function 263

SPI_Data_Write_Read_Helper_Commit 270

SPI_Data_Write_Read_Helper_Commit function 270

SPI_Data_Write_Read_Helper_Read 271

SPI_Data_Write_Read_Helper_Read function 271

SPI_Data_Write_Read_Helper_Write 273

SPI_Data_Write_Read_Helper_Write function	273	STOPWATCH_ERROR	369
spi_dataset	311	STOPWATCH_ERROR macro	369
spi_dataset structure	311	STOPWATCH_HANDLE_TYPE	369
SPI_FIFO_Read	276	STOPWATCH_HANDLE_TYPE macro	369
SPI_FIFO_Read function	276	STOPWATCH_INACTIVE	370
SPI_FIFO_Write	278	STOPWATCH_INACTIVE macro	370
SPI_FIFO_Write function	278	StopWatch_Initialization	291
SPI_ID_Get	279	StopWatch_Initialization function	291
SPI_ID_Get function	279	StopWatch_Open	291
SPI_ID_Get_Helper	280	StopWatch_Open function	291
SPI_ID_Get_Helper function	280	STOPWATCH_PAUSE	370
SPI_IsNotPresent	280	STOPWATCH_PAUSE macro	370
SPI_IsNotPresent function	280	StopWatch_Process	291
SPI_REGISTER_SET_DEFINITION	368	StopWatch_Process function	291
SPI_REGISTER_SET_DEFINITION macro	368	STOPWATCH_QUANTITY	370
SPI_Report_Configuration_Text	281	STOPWATCH_QUANTITY macro	370
SPI_Report_Configuration_Text function	281	STOPWATCH_RESET	370
SPI_Report_Status_Text	282	STOPWATCH_RESET macro	370
SPI_Report_Status_Text function	282	StopWatch_Set	291
spi_status	311	StopWatch_Set function	291
spi_status structure	311	STOPWATCH_START	371
SPI_Status_Read	283	STOPWATCH_START macro	371
SPI_Status_Read function	283	STOPWATCH_STATE_TYPE	371
SPI_Status_Read_FIFO_Is_Not_Empty	284	STOPWATCH_STATE_TYPE macro	371
SPI_Status_Read_FIFO_Is_Not_Empty function	284	StopWatch_Termination	292
SPI_Status_Read_FIFO_Status	284	StopWatch_Termination function	292
SPI_Status_Read_FIFO_Status function	284	StopWatch_TimeStamp_String	292
SPI_Status_Write	286	StopWatch_TimeStamp_String function	292
SPI_Status_Write function	286	STOPWATCH_TIMEVAL_TYPE	321
SPI_Status_Write_FIFO_Is_Full	287	STOPWATCH_TIMEVAL_TYPE type	321
SPI_Status_Write_FIFO_Is_Full function	287	StopWatch_Value	292
SPI_Status_Write_FIFO_Is_Not_Empty	288	StopWatch_Value function	292
SPI_Status_Write_FIFO_Is_Not_Empty function	288	strcmpl	371
SPI_Status_Write_FIFO_Status	288	strcmpl macro	371
SPI_Status_Write_FIFO_Status function	288	String_To_Bool	293
stopwatch.h	580	String_To_Bool function	293
StopWatch_Close	290	Structs, Records, Enums	295
StopWatch_Close function	290	SVN_REV	372

SVN_REV macro 372
svn_revision_string 337
svn_revision_string variable 337

T

TARGET_CPU_INTEL_386 372
TARGET_CPU_INTEL_386 macro 372
Termination 293
Termination function 293
TEST_STATE_ENUM 317
TEST_STATE_ENUM enumeration 317
Time_Current_String 294
Time_Current_String function 294
timeval 312
timeval structure 312
true 372
true macro 372
Types 318

U

uint16_t 321
uint16_t type 321
UINT32_MAX 372
UINT32_MAX macro 372
uint32_t 321
uint32_t type 321
uint8_t 321
uint8_t type 321

V

Variables 322

W

Welcome 1