AMIXXX Device Driver Specifications

Aichi Steel Corporation

Version: 1.3.0 (2013/03/27)

Revision

Version	Date	Contents	Remarks
1.0	2012/06/08	Original	
1.01	2012/07/18	Added Functions: AMI_SetDirection	
1.02	2012/07/26	Deleted-function: AMI_SetDirection AMI_SetInterferenceOffset Added Functions: AMI_SetSoftIron	
1.03	2012/08/01	Added Functions: AMI_SelfTest	
1.04	2012/08/09	Added Functions: AMI_SetDirection AMI_GetDirection	
1.05	2012/10/17	Added Functions: AMI_GetSoftIron	
1.2.0	2012/12/06	Version number change only.	
1.3.0	2013/03/27	Added AMI603	

Table of Contents

1 Outline

This document is the Specifications relating to the AMI306/AMI307/AMI603 (below replace AMIXXX) device driver.

1.1 System Structure

The following is the system structure diagram:

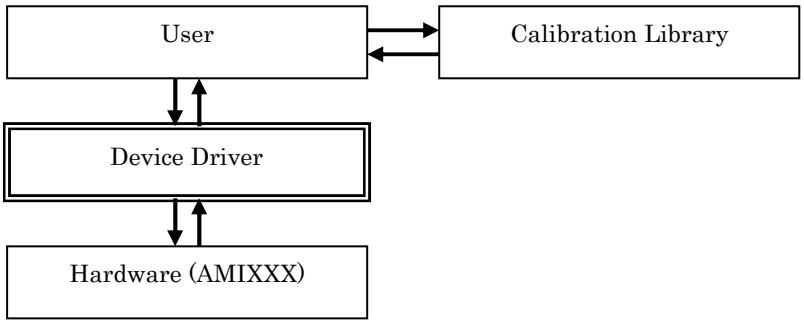


Figure 1-1. System Structure

① User

User obtains raw magnetic data from the Device Driver and passes it to the Calibration Library to obtain calibrated magnetic data.

② Calibration Library

Calibration Library calibrates raw magnetic data obtained from the Device Driver via the User and passes calibrated magnetic values to the User.

③ Device Driver

Device Driver controls the AMIXXX hardware with I2C and obtains raw magnetic data. The API Group used by the User is installed here.

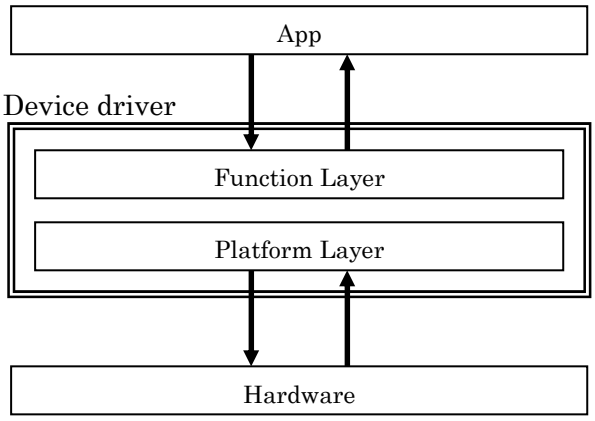
④ Hardware

Hardware is the AMIXXX sensor.

This document describes the “Device Driver” shown in the above diagram.

1.2 Device Driver Structure

The Device Driver is constructed of the following layers:



(1) Platform Layer

The Platform Layer refers to the parts which are reliant on the Platform and are installed to match the Platform.

(2) Function Layer

The Function Layer refers to the parts where the Device Driver functions are installed and is not dependant on the Platform.

2 API Specification

2.1 Platform Layer Function Specifications

The functions specific to the Platform need to be installed on the User side.

(1) Microsecond Sleep

void AMI_udelay(u32 usec);			
	Type Name	Variable Name	Details
Argument	u32	usec	Microseconds
Return Vaue	None	—	—

Pauses for the indicated period of microseconds only

(2) Millisecond Sleep

void AMI_mdelay(u32 usec);			
	Type Name	Variable Name	Details
Argument	ami_uint8	msec	Milliseconds
Return Value	None	—	—

Pauses for the indicate period of milliseconds only

(3) Obtain DRDY Status

int AMI_DRDY_Value(void);			
	Type Name	Variable Name	Details
Argument	None	—	—
Return Value	ami_uint8	—	0 : DRDY low 1 : DRDY high

Obtains DRDY status

(4) I2C Transmission

int AMI_i2c_send(void *i2c, u8 adr, u8 len, u8 *buf);			
	Type Name	Variable Name	Details
Argument	void *	i2c	i2c handle
	u8	adr	Register address
	u8	len	Data size
	u8*	buf	Address of sent data
Return Value	ami_uint8	—	Result 0: Normal, other than 0: abnormal

Sends data over I2C

i2c handle is the value passed by the Argument of 「AMI_InitDriver()」

(5) I2C Transmission

int AMI_i2c_recv(void *i2c, u8 adr, u8 len, u8 *buf);			
	Type Name	Variable	Details
Argument	void *	i2c	i2c handle
	u8	adr	Register address
	u8	len	Data size
	u8*	buf	Address of received data
Return Value	ami_uint8	—	Result 0: Normal, other than 0: abnormal

Receives data over I2C

i2chandle is the value passed by the Argument of 「AMI_InitDriver()」

(6) Log output

void AMI_LOG(const char *fmt, ...);			
	Type Name	Variable Name	Details
Argument	const char *	fmt	Output format
	…	—	Output data
Return Value	None	—	—

Outputs log

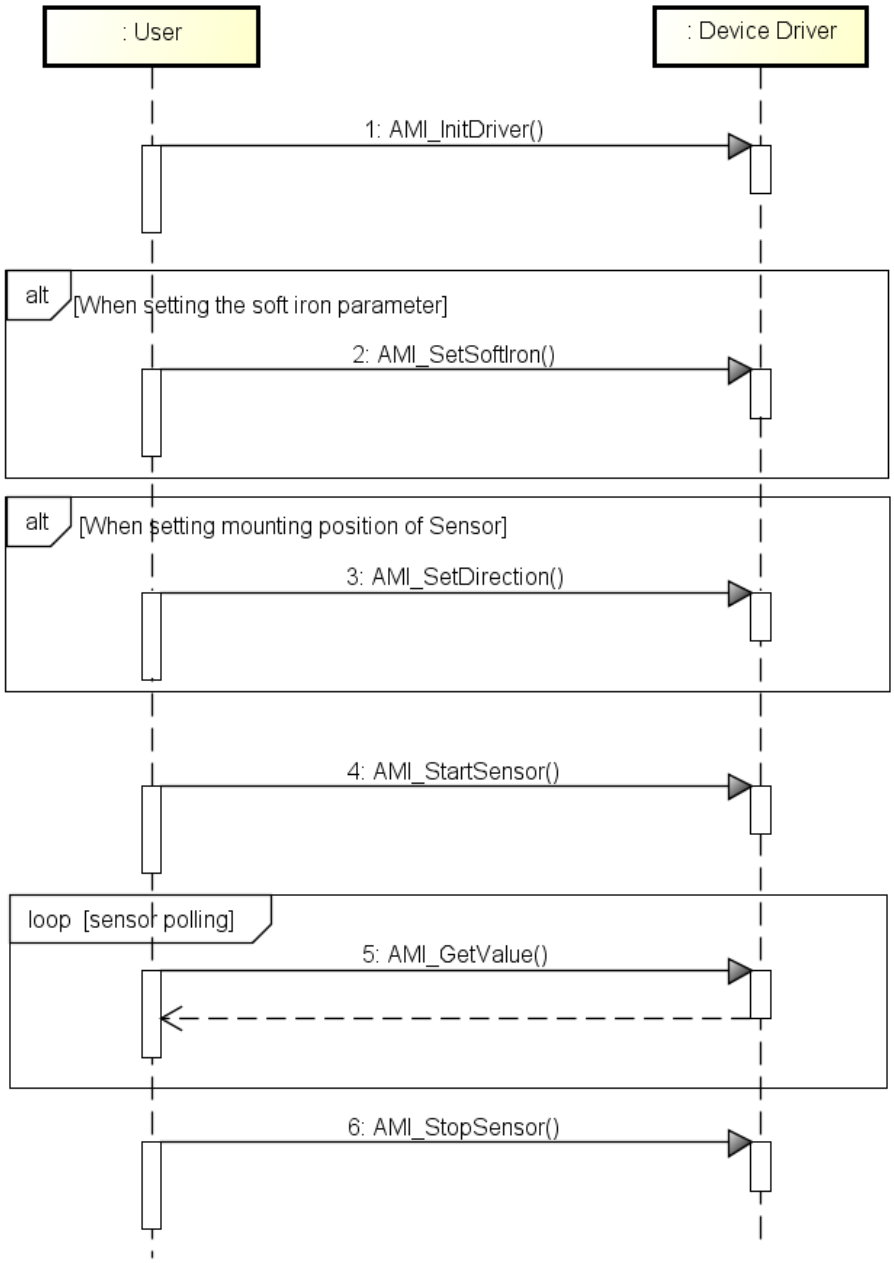
(6) Output debug log

void AMI_DLOG(const char *fmt, ...);			
	Type Name	Variable Name	Details
Argument	const char *	fmt	Output format
	…	—	Output data
Return Value	None	—	—

Outputs log for debugging

2.2 Function Layer Functions Specifications

【Basic Sequence】



(1) Device Driver Initialization

void *AMI_InitDriver(void *i2c_handle);			
	Type name	Variable Name	Details
Argument	void *	i2c_handle	i2c handle
Return Value	void *	handle	Device Driver handle

Initializes Device Driver

「i2c handle」 is the value passed by the Argument of the I2C transmission Variable (AMI_i2c_send/ AMI_i2c_recv)

(2) Starting Sensor

int AMI_StartSensor(void *handle);			
	Type Name	Variable Name	Detail
Argument	void *	handle	Device Driver handle (AMI_InitDriver() return value)
Return Value	int	—	Result 0: Normal, other than 0: abnormal

Begins sensor with sensor in Force Mode

(3) End Sensor

int AMI_StopSensor(void *handle);			
	Type Name	Variable Name	Details
Argument	void *	handle	Device Driver handle (AMI_InitDriver() return value)
Return Value	int	—	Result 0: Normal, other than 0: abnormal

Stops sensor (Sensor enters Standby Mode)

(4) Obtain Data

int AMI_GetValue(void *handle, struct ami_sensor_value *val);			
	Type Name	Variable Name	Details
Argument	void *	handle	Device Driver handle (AMI_InitDriver() return value)
	struct ami_sensor_value*	val	Data
Return Value	int	—	Result 0: Normal, other than 0: abnormal

Obtains magnetic data

(5) Offset Adjustment

int AMI_SearchOffset(void *handle);			
	Type Name	Variable Name	Details
Argument	void *	handle	Device Driver handle (AMI_InitDriver() return value)
Return Value	int	—	Result 0: Normal, other than 0: abnormal

Adjusts the sensor offset

Call this function when forcibly adjusting the offset.

(6) Setting offset

int AMI_WriteOffset(void *handle, u8 offset[3]);			
	Type Name	Variable Name	Details
Argument	void *	handle	Device Driver handle (AMI_InitDriver() return value)
	u8	offset[[3]	Offset [0]:X-axis、[1]:Y-axis、[2]:Z-axis
Return Value	int	—	Result 0: Normal, other than 0: abnormal

Sets the sensor offset

Call this function to forcibly set the offset.

(7) Obtaining Offset

int AMI_ReadOffset(void *handle, u8 offset[3]);			
	Type Name	Variable Name	Details
Argument	void *	handle	Device Driver handle (AMI_InitDriver() return value)
	u8	offset[[3]	Offset [0]:X-axis, [1]:Y-axis, [2]:Z-axis
Return Value	int	—	Result 0: Normal, other than 0: abnormal

Obtains the sensor offset

(8) Setting Axial soft iron parameter

int AMI_SetSoftIron(void *handle, s16 si[9]);			
	Type Name	Variable Name	Details
Argument	void *	handle	Device Driver handle (AMI_InitDriver() return value)
	S16	si[9]	Axial soft iron parameter value [0]:XX-axes, [1]:XY-axes, [2]:XZ-axes, [3]:YX-axes, [4]:YY-axes, [5]:YZ-axes, [6]:ZX-axes, [7]:ZY-axes [8]:ZZ-axes
Return Value	int	—	Result 0: Normal, other than 0: abnormal

Sets the value to soft iron parameter

(9) Getting Axial soft iron parameter

int AMI_GetSoftIron(void *handle, s16 si[9]);			
	Type Name	Variable Name	Details
Argument	void *	handle	Device Driver handle (AMI_InitDriver() return value)
	S16	si[9]	Axial soft iron parameter value [0]:XX-axes, [1]:XY-axes, [2]:XZ-axes, [3]:YX-axes, [4]:YY-axes, [5]:YZ-axes, [6]:ZX-axes, [7]:ZY-axes [8]:ZZ-axes
Return Value	int	—	Result 0: Normal, other than 0: abnormal

Gets the value to soft iron parameter

(10) Setting mounting Position of Sensor

int AMI_SetDirection(void *handle, s16 dir, s16 polarity);			
	Type Name	Variable Name	Details
Argument	void *	handle	Device Driver handle (AMI_InitDriver() return value)
	s16	dir	Sensor axial information relative to device axes.
	s16	polarity	Sensor polarity information relative to device poles.
Return Value	int	—	Result 0: Normal, other than 0: abnormal

See the "【Choosing Mounting Position of Sensor】" page.

(11) Getting mounting Position of Sensor

int AMI_GetDirection(void *handle, s16 *dir, s16 *polarity);			
	Type Name	Variable Name	Details
Argument	void *	handle	Device Driver handle (AMI_InitDriver() return value)
	s16	*dir	Sensor axial information relative to device axes.
	s16	*polarity	Sensor polarity information relative to device poles.
Return Value	int	—	Result 0: Normal, other than 0: abnormal

See the "【Choosing Mounting Position of Sensor】" page.

(12) Self Test (AMI306 only)

int AMI_SelfTest (void *i2c_handle);			
	Type Name	Variable Name	Details
Argument	void *	i2c_handle	i2c handle
Return Value	int	—	Self test result. See "2.4.2 List of self test result."

Execute self test.

「i2c handle」 is the value passed by the Argument of the I2C transmission Variable (AMI_i2c_send/ AMI_i2c_recv)

2.3 Structural Definitions

2.3.1 Magnetic Information (In the case of AMI30X)

struct ami_sensor_value		
Variable Name	**Type**	**Details**
mag[3]	signed short	Magnetism Value (Unit: mGauss) [0]:X-Axis, [1]:Y-Axis, [2]:Z-Axis

2.3.2 Magnetic and Acceleration Information (In the case of AMI60X)

struct ami_sensor_value		
Variable Name	**Type**	**Details**
mag[3]	signed short	Magnetism Value (Unit: mGauss) [0]:X-Axis, [1]:Y-Axis, [2]:Z-Axis
acc[3]	signed short	Acceleration Value (Unit: mG) [0]:X-Axis, [1]:Y-Axis, [2]:Z-Axis

2.4 List of Errors

2.4.1 List of Errors

Type of Error	Error Code	Details
Normal	0	Normal operation
Parameter error	-1	Argument error or other input value anomaly
Sequence error	-2	Sequence error
Transmission error	-3	Transmission error
System error	-10	Hardware anomaly or other system error
Other error	-99	Other error

2.4.2 List of self test result

Type of Error	Result code	Details
AMI_ST_OK	0	Self test OK
AMI_ST_ERR_COMMUNICATION	1	Communication Error
AMI_ST_ERR_OTP	2	OTP Error
AMI_ST_ERR_TEMP_SENSOR	3	Temp sensor Error
AMI_ST_ERR_MI_ELEMENT	4	MI element Error
AMI_ST_ERR_DIGITAL_CIRCUIT	5	Digital circuit Error

3 Source Code Revisions

Revision of the Source Code and Build Options to match the Platform will be required.

Depending on where the sensor is mounted, revision of the Source Code and Build Options will be necessary.

3.1 Source Code

Target File: AMIXXX_hw.h
Target Item: ①AMI_DIR: Set the location the sensor is mounted on the device (See 「Choosing Mounting Position of Sensor」) ②AMI_POLARITY: Set the polarity of the sensor in relation to the device (See「Choosing Mounting Position of Sensor」)

3.2 Build Option

①DEBUG_LOG: When outputting the debug log ②USE_DRDY_PIN: When using the DRDY pin ③DELAY_MILLI_SEC: When the udelay() function does not respond

【Choosing Mounting Position of Sensor】

Depending on the mounting position, the device axes and sensor axes may not match. The axial information based on the device can be changed using AMI_DIR and AMI_POLARITY.

Item	Detail
AMI_DIR	Sensor axial information relative to device axes.
AMI_POLARITY	Sensor polarity information relative to device poles.

1. AMI_DIR

Bit	7	6	5	4	3	2	1	0
Detail	—	—	Device X-axis		Device Y-axis		Device Z-axis	

The information for each sensor axis is set with the following bits:

Axis	Bit 1	Bit 0
X	0	0
Y	0	1
Z	1	0

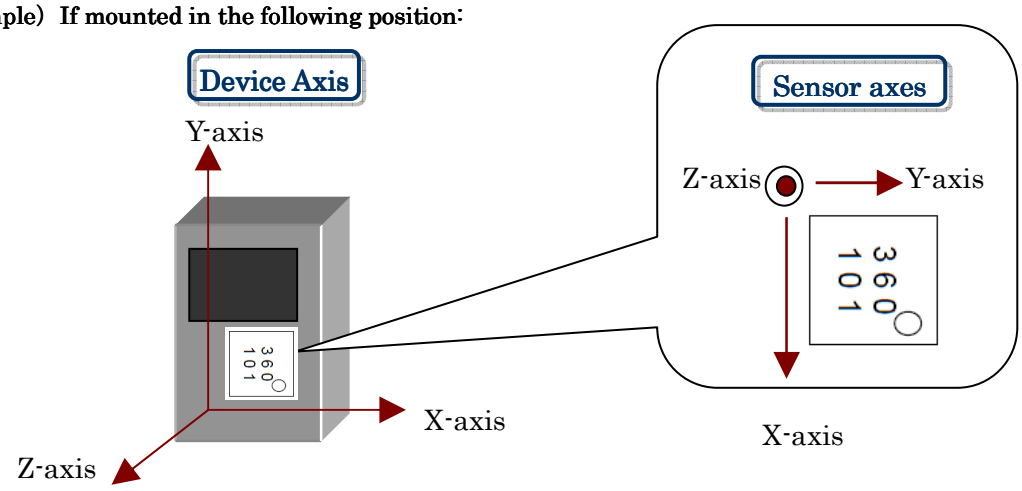
2. AMI_POLARITY

The following are the polarities of each axis as seen from the right hand polarity of the device axes (Appendix-1 Definitions of Device axes)

Bit	7	6	5	4	3	2	1	0
Detail	—	—	—	—	—	X-axis polarity	Y-axis polarity	Z-axis polarity

If positive set 「1」, if negative set「0」.

Example) If mounted in the following position:



Variable	Value	Detail
AMI_DIR	18(0x12)	Device X-axis=sensor Y-axis(01), Device Y-axis=sensor X-axis (00), Device Z-axis=sensor Z-axis(10) bit (0 0 0 1 0 0 1 0)
AMI_POLARITY	5(0x5)	Device X-axis and sensor Y-axis are positive (1), Device Y-axis and sensor X-axis are negative (0), device Z-axis and sensor Z-axis are positive (1) bit (0 0 0 0 0 1 0 1)