AMI6XX Device Driver Specifications

愛知製鋼株式会社

Version: 1.4.0 (2013/05/24)

Revision

Version	Date	Contents	Remarks
1.0	2012/06/08	新規作成	
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	
1.4.0	2013/05/24	Added pedometer function.	

Table of Contents

1 概要

本書は、AMI603(以下 AMIXXX)のデバイスドライバに関する仕様書である。

1.1 システム構成

以下にシステム構成図を記す。

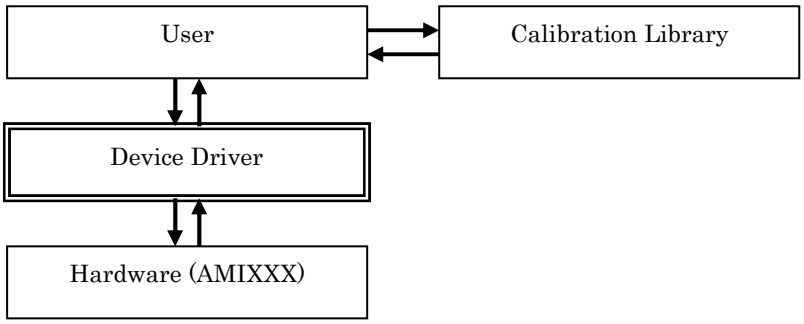


Figure 1-1. システム構成図

① User

Device Driver より磁気の生データを取得して、Calibration Library にデータを渡し、校正された磁気データを取得する。

② Calibration Library

User 経由で Device Driver から取得した磁気の生値を校正し、校正された磁気の値を User に渡す。

③ Device Driver

AMIXXX のハードウェアを I2C 通信で制御し磁気の生データを取得する。User で使用する API 群が実装されている。

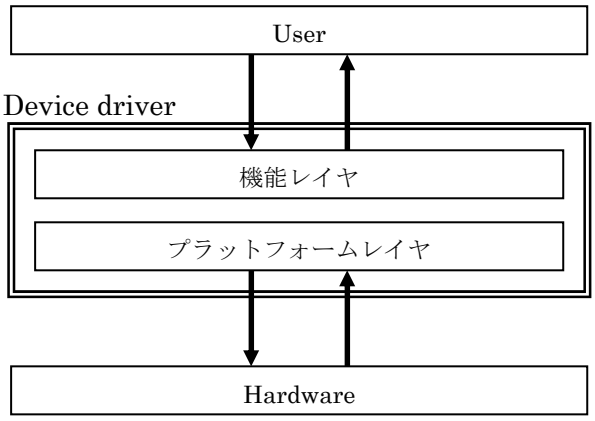
④ Hardware

AMIXXX センサー。

本書では図中”Device Driver”について説明する。

1.2 Device Driver 構成

Device Driver は以下の階層で構成される。



(1)プラットフォームレイヤ

プラットフォームに依存した部分で、プラットフォームに合わせた実装を行う。

(2)機能レイヤ

Device Driver の機能を実装した部分で、プラットフォームに依存しない。

2 API 仕様

2.1 プラットフォームレイヤ 関数仕様

プラットフォームに対応した関数をユーザ側で実装する必要がある。

(1) マイクロ秒スリープ

void AMI_udelay(u32 usec);			
	型名	変数名	説明
引　数	u32	usec	マイクロ秒
戻り値	なし	－	－

指定のマイクロ秒の間だけ休止する。

(2) ミリ秒スリープ

void AMI_mdelay(u32 usec);			
	型名	変数名	説明
引　数	ami_uint8	msec	ミリ秒
戻り値	なし	－	－

指定のミリ秒の間だけ休止する。

(3) DRDY 状態取得

int AMI_DRDY_Value(void);			
	型名	変数名	説明
引　数	なし	－	－
戻り値	ami_uint8	－	0 : DRDY low 1 : DRDY high

DRDY の状態を取得する。

(4) I2C 送信

int AMI_i2c_send(void *i2c, u8 adr, u8 len, u8 *buf);			
	型名	変数名	説明
引　数	void *	i2c	i2c ハンドル
	u8	adr	レジスタアドレス
	u8	len	データ長
	u8*	buf	送信データアドレス
戻り値	ami_uint8	－	結果 0:正常, 0 以外:異常

I2C でデータを送信する。

i2c ハンドルは「AMI_InitDriver() (*1)」の引数で渡した値。

(5) I2C 送信

int AMI_i2c_recv(void *i2c, u8 adr, u8 len, u8 *buf);			
	型名	変数名	説明
引　数	void *	i2c	i2c ハンドル
	u8	adr	レジスタアドレス
	u8	len	データ長
	u8*	buf	受信データアドレス
戻り値	ami_uint8	－	結果 0:正常, 0 以外:異常

I2C でデータを受信する。

i2c ハンドルは「AMI_InitDriver() (*1)」の引数で渡した値。

(*1)　2.2 機能レイヤ 関数仕様 (1) 参照

(6) ログ出力

void AMI_LOG(const char *fmt, ...);			
	型名	変数名	説明
引　数	const char *	fmt	出力フォーマット
	…	－	出力データ
戻り値	なし	－	－

ログを出力する。

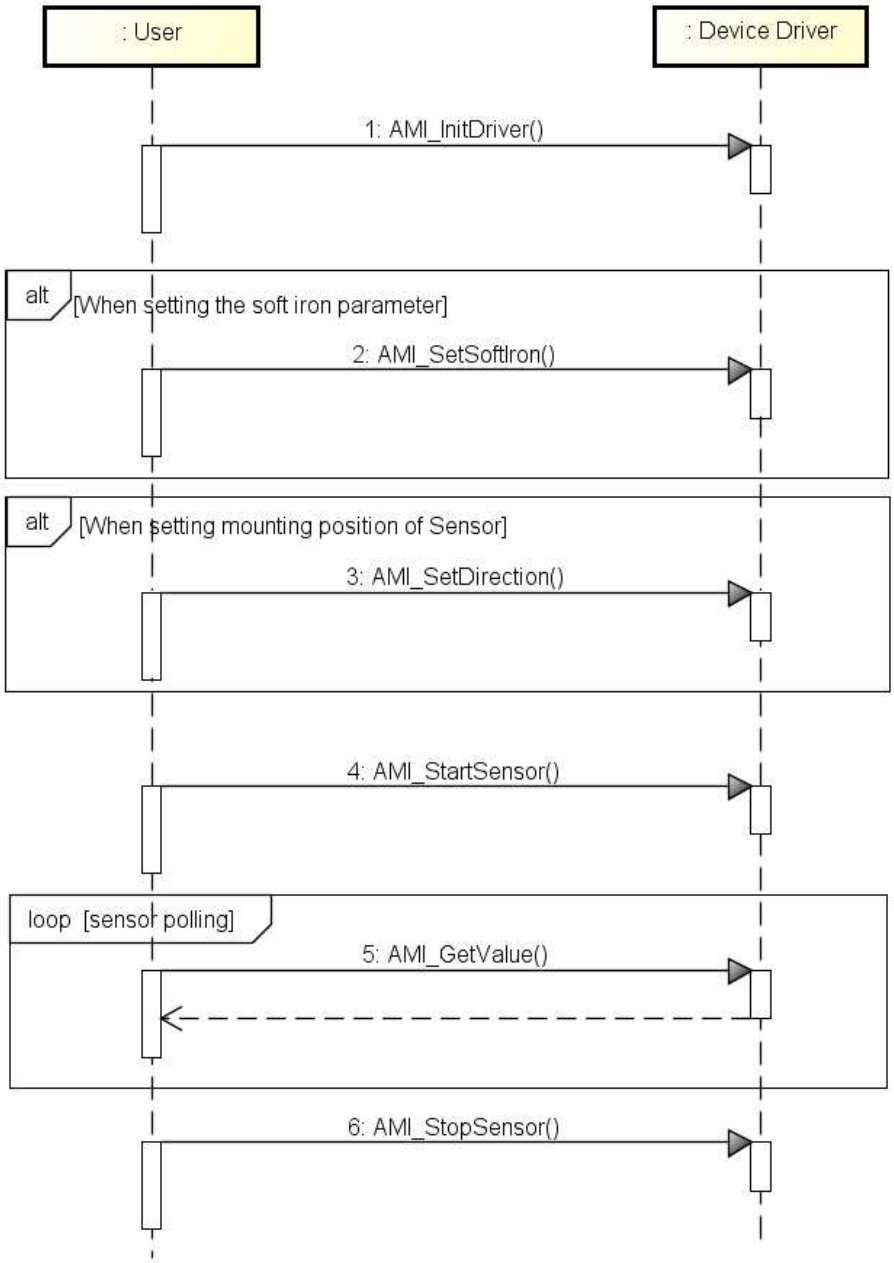
(6) デバッグログ出力

void AMI_DLOG(const char *fmt, ...);			
	型名	変数名	説明
引　数	const char *	fmt	出力フォーマット
	…	－	出力データ
戻り値	なし	－	－

デバッグ用のログを出力する。

2.2 機能レイヤ 関数仕様

【基本シーケンス】



(1) Device Driver 初期化

void *AMI_InitDriver(void *i2c_handle);			
	型名	変数名	説明
引　数	void *	i2c_handle	i2c ハンドル
戻り値	void *	handle	Device Driver ハンドル

Device Driver を初期化する。

「i2c ハンドル」は I2C 送信/受信関数(AMI_i2c_send/ AMI_i2c_recv)の引数で渡す値。

(2) センサー開始

int AMI_StartSensor(void *handle, int mode, int interval);			
	型名	変数名	説明
引　数	void *	handle	Device Driver ハンドル(AMI_InitDriver()の戻り値)
	int	mode	0:normal 1:force
	int	interval	normal-mode interval (20,40,60,80,100)
戻り値	int	－	結果 0:正常, 0 以外:異常

センサーをフォースモード、もしくはノーマルモードで開始する。

Force-mode でセンサーを起動している時は、歩数計は起動できない。

Normal-mode で interval が 60, 80, 100 の時、歩数計を起動した場合、歩数計起動中は強制的に 40msec で動作させる。

(3) センサー終了

int AMI_StopSensor(void *handle);			
	型名	変数名	説明
引　数	void *	handle	Device Driver ハンドル(AMI_InitDriver()の戻り値)
戻り値	int	－	結果 0:正常, 0 以外:異常

センサーが終了する。(センサーをスタンバイモードにする。)

(4) データ取得

int AMI_GetValue(void *handle, struct ami_sensor_value *val);			
	型名	変数名	説明
引　数	void *	handle	Device Driver ハンドル(AMI_InitDriver()の戻り値)
	struct ami_sensor_value*	val	データ
戻り値	int	―	結果 0:正常, 0 以外:異常

磁気データを取得する。

(5) オフセット調整

int AMI_SearchOffset(void *handle);			
	型名	変数名	説明
引　数	void *	handle	Device Driver ハンドル(AMI_InitDriver()の戻り値)
戻り値	int	―	結果 0:正常, 0 以外:異常

センサーのオフセットを調整する。

強制的にオフセットを調整する場合に、この関数を呼び出す。

(6) オフセット設定

int AMI_WriteOffset(void *handle, u8 offset[3]);			
	型名	変数名	説明
引　数	void *	handle	Device Driver ハンドル(AMI_InitDriver()の戻り値)
	u8	offset[[3]	オフセット [0]:X 軸、[1]:Y 軸、[2]:Z 軸
戻り値	int	―	結果 0:正常, 0 以外:異常

センサーのオフセットを設定する。

強制的にオフセットを設定する場合に、この関数を呼び出す。

(7) オフセット取得

int AMI_ReadOffset(void *handle, u8 offset[3]);			
	型名	変数名	説明
引　数	void *	handle	Device Driver ハンドル(AMI_InitDriver()の戻り値)
	u8	offset[[3]	オフセット [0]:X 軸、[1]:Y 軸、[2]:Z 軸
戻り値	int	－	結果 0:正常, 0 以外:異常

センサーのオフセットを取得する。

(8) ソフトアイロン補正値設定

int AMI_SetSoftIron(void *handle, s16 si[9]);			
	型名	変数名	説明
引　数	void *	handle	Device Driver ハンドル(AMI_InitDriver()の戻り値)
	S16	si[9]	ソフトアイロン補正値 [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
戻り値	int	－	結果 0:正常, 0 以外:異常

ソフトアイロンを補正する値を設定する。

(9) ソフトアイロン補正値取得

int AMI_SetGoftIron(void *handle, s16 si[9]);			
	型名	変数名	説明
引　数	void *	handle	Device Driver ハンドル(AMI_InitDriver()の戻り値)
	S16	si[9]	ソフトアイロン補正値 [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
戻り値	int	－	結果 0:正常, 0 以外:異常

ソフトアイロンを補正する値を取得する。

(10) センサー取り付け位置情報設定

int AMI_SetDirection(void *handle, s16 dir, s16 polarity);			
	型名	変数名	説明
引　数	void *	handle	Device Driver ハンドル(AMI_InitDriver()の戻り値)
	s16	dir	機器軸に対する磁気センサーの軸情報。
	s16	polarity	機器に対する磁気センサーの極性情報
戻り値	int	－	結果 0:正常, 0 以外:異常

【センサー取り付け位置設定方法】参照

(11) センサー取り付け位置情報取得

int AMI_GetDirection(void *handle, s16 *dir, s16 *polarity);			
	型名	変数名	説明
引　数	void *	handle	Device Driver ハンドル(AMI_InitDriver()の戻り値)
	s16	*dir	機器軸に対する磁気センサーの軸情報。
	s16	*polarity	機器に対する磁気センサーの極性情報
戻り値	int	－	結果 0:正常, 0 以外:異常

【センサー取り付け位置設定方法】参照

(12) 歩数計開始

int AMI_StartPedometer(void *handle);			
	型名	変数名	説明
引　数	void *	handle	Device Driver ハンドル(AMI_InitDriver()の戻り値)
戻り値	int	－	結果 0:正常, 0 以外:異常

歩数計を開始する。

Force-mode でセンサーを起動している時は、歩数計は起動できない。

Normal-mode で interval が 60, 80, 100 の時、歩数計を起動した場合、歩数計起動中は強制的に 40msec で動作させる。

(13) 歩数計終了

int AMI_StopPedometer (void *handle);			
	型名	変数名	説明
引　数	void *	handle	Device Driver ハンドル(AMI_InitDriver()の戻り値)
戻り値	int	－	結果 0:正常, 0 以外:異常

歩数計を停止する。

AMI_StartPedometer で interval が変更されていた場合、元の値に戻ります。

(14) 歩数計情報取得

int AMI_GetPedometer(void *handle, struct ami_pedo_status *stat);			
	型名	変数名	説明
引　数	void *	handle	Device Driver ハンドル(AMI_InitDriver()の戻り値)
	struct ami_pedo_status	*stat	歩数計情報
戻り値	int	－	結果 0:正常, 0 以外:異常

歩数計情報を取得する。

(15) 歩数計情報クリア

int AMI_ResetPedometer(void *handle);			
	型名	変数名	説明
引　数	void *	handle	Device Driver ハンドル(AMI_InitDriver()の戻り値)
戻り値	int	－	結果 0:正常, 0 以外:異常

歩数計情報をクリアする。

2.3 構造体定義

2.3.1 磁気、加速度情報

struct ami_sensor_value		
変数名	**型**	**説明**
mag[3]	signed short	磁気の値(単位:mGauss) [0]:X 軸、[1]:Y 軸、[2]:Z 軸
acc[3]	signed short	加速度の値(単位:mG) [0]:X 軸、[1]:Y 軸、[2]:Z 軸

2.3.2 歩数計情報

struct ami_pedo_status		
変数名	**型**	**説明**
count	unsigned long	歩数
time	unsigned long	歩行時間 (秒)
stat	unsigned long	歩行状態 (0:停止状態、1:歩行チェック状態、2:歩行状態)

2.4 エラーー一覧

2.4.1 エラーー一覧

エラーの種類	エラーコード	説明
正常	0	正常動作
パラメータエラー	-1	引数エラーなど入力値異常
シーケンスエラー	-2	シーケンスエラー
通信エラー	-3	通信エラー
システムエラー	-10	ハードウェア異常等のシステム的なエラー
その他のエラー	-99	その他のエラー

2.4.2 セルフテスト結果

種類	コード	説明
AMI_ST_OK	0	OK
AMI_ST_ERR_COMMUNICATION	1	通信エラー
AMI_ST_ERR_OTP	2	OTP エラー
AMI_ST_ERR_TEMP_SENSOR	3	温度センサーエラー
AMI_ST_ERR_MI_ELEMENT	4	MI エレメントエラー
AMI_ST_ERR_DIGITAL_CIRCUIT	5	デジタル回路エラー

3　ソースコードの修正

プラットフォームに対応したソースコード及び、ビルドオプションの修正が必要になります。

また、センサーの取り付け位置により、ソースコード及び、ビルドオプションの修正が必要になります。

3.1　　ソースコード

対象ファイル：AMIXXX_hw.h
対象項目： ①AMI_DIR：　　端末に磁気センサーを取り付けた位置を設定（「センサー取り付け位置設定方法」参照） ②AMI_POLARITY：　端末に対する磁気センサーの極性を設定（「センサー取り付け位置設定方法」参照）

3.2　　ビルドオプション

①DEBUG_LOG：　　デバッグログを出力する場合 ②USE_DRDY_PIN：　DRDY ピンを使用する場合 ③DELAY_MILLI_SEC：udelay()関数が対応していない場合

【センサー取り付け位置設定方法】

センサーの取り付け位置により機器軸とセンサー軸は一致しません。AMI_DIR と AMI_POLARITY により、機器を基準とした軸の情報へ変換する。

項目	説明
AMI_DIR	機器軸に対する磁気センサーの軸情報。
AMI_POLARITY	機器に対する磁気センサーの極性情報

1．AMI_DIR

Bit	7	6	5	4	3	2	1	0
説明	―	―	機器 X 軸		機器 Y 軸		機器 Z 軸	

各センサーの軸情報は以下のビットを設定する。

軸	Bit 1	Bit 0
X	0	0
Y	0	1
Z	1	0

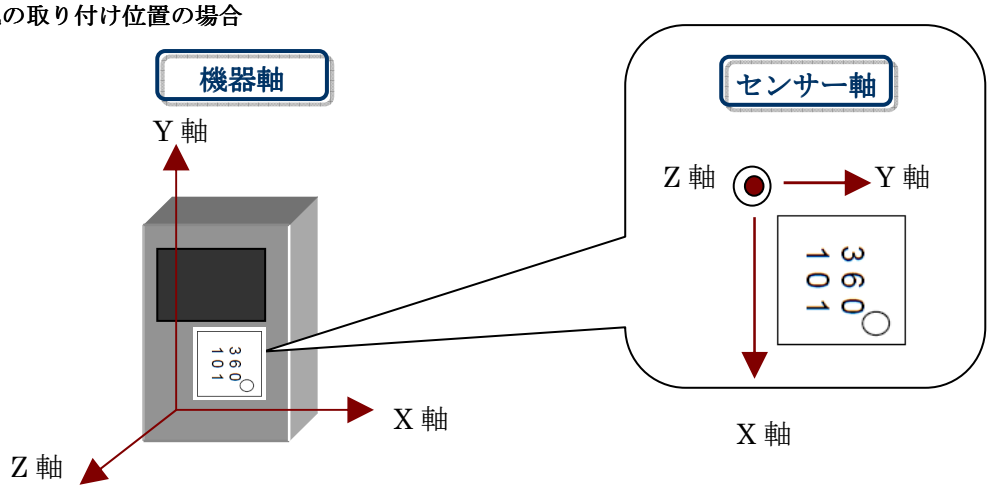
2．AMI_POLARITY

機器軸の右手系極性(Appendix-1 の「機器の軸定義」)からみた各軸の極性を表す。

Bit	7	6	5	4	3	2	1	0
説明	―	―	―	―	―	X 軸極性	Y 軸極性	Z 軸極性

正方向なら「1」、逆方向なら「0」を設定する。

例）下記の取り付け位置の場合



変数	値	内容
AMI_DIR	18(0x12)	機器 X 軸=センサーY 軸(01)、機器 Y 軸=センサーX 軸(00)、 機器 Z 軸=センサーZ 軸(10) bit （ 0 0 0 1 0 0 1 0 ）
AMI_POLARITY	5(0x5)	機器 X 軸とセンサーY 軸は正方向(1)、機器 Y 軸とセンサーX 軸は逆方向(0)、 機器 Z 軸とセンサーZ 軸は正方向(1) bit （ 0 0 0 0 0 1 0 1 ）