



RF/MULTI(センサ)の動作説明

フリースケール・セミコンダクタ・ジャパン
株式会社




Nov 2014

External Use

Freescale, the Freescale logo, Altivec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, Qorivva, SafeAssure, the SafeAssure logo, StarCore, Symphony and VortiQa are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Airfast, BeeKit, BeeStack, CoreNet, Flexis, Layerscape, MagniV, MXC, Platform in a Package, QorIQ Qonverge, QUICC Engine, Ready Play, SMARTMOS, Tower, TurboLink, UMEMS, Vybrid and Xtrinsic are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. © 2014 Freescale Semiconductor, Inc.



- 
1. センサの接続及びセンサ説明
 2. Softwareの実装準備
 3. IIC driver実装
 4. 各センサーのconfigurationの実装
 1. FXAS2100: 3軸ジャイロセンサ
 2. MMA9553: インテリジェント歩数計(MCU+加速度)
 3. MPL3115A2: 気圧センサ
 4. MMA8652: 3軸加速度センサ
 5. MAG3110: 3軸磁気センサ



センサの接続及びセンサ説明

RFボードとFRDM-FXS-MULTIボードの接続

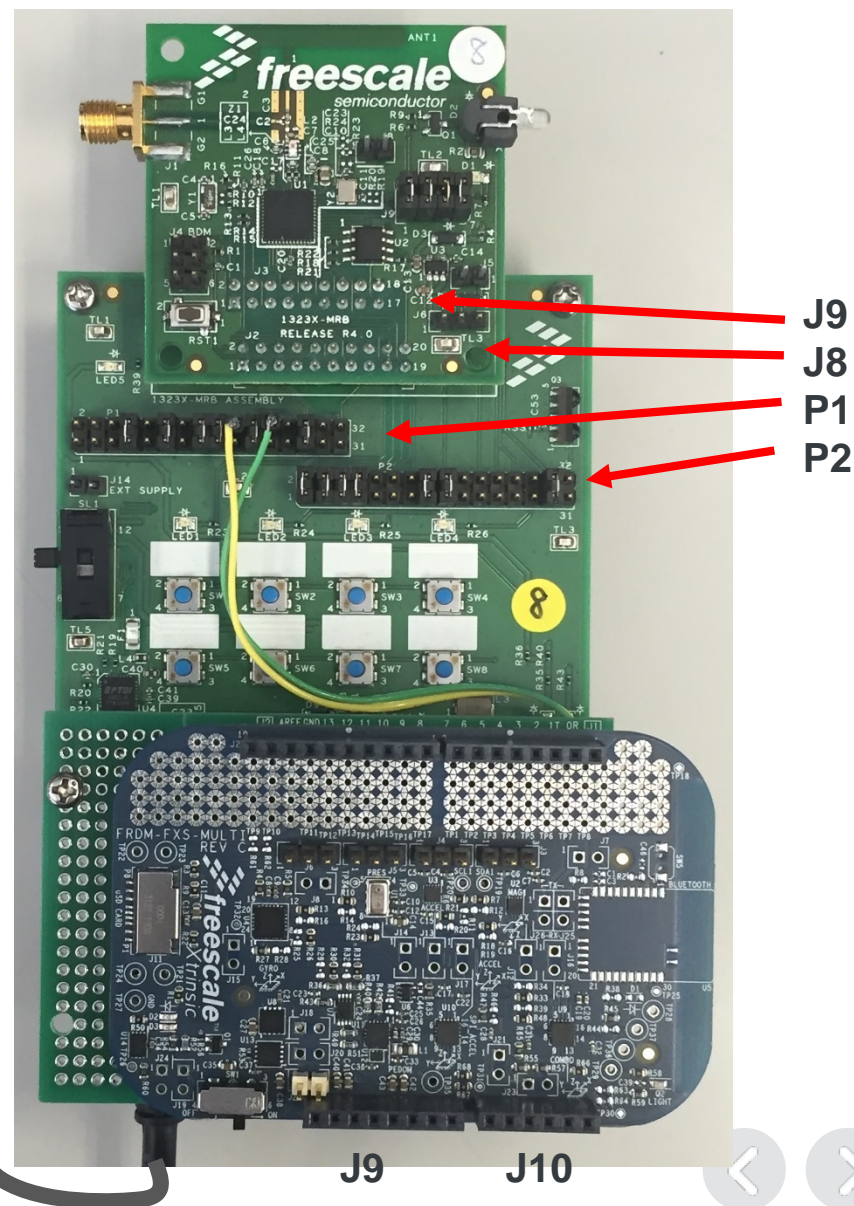
RFとMULTIセンサ・ボード間の結線表

Signal	RF(REM board)	MULTI
3V	J8-1	J9-4
GND	J8-3	J9-7
SDA	P1-20	J10-5
SCL	P1-24	J10-6



センサ・データは通常の
ターミナルソフトで確認
可能。

115.2kpbs



Terminal設定

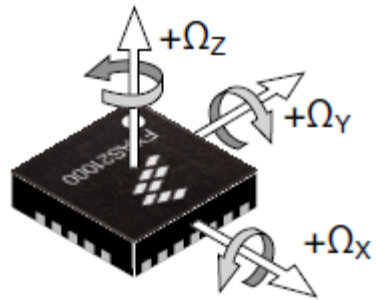
era Term: シリアルポート 設定

ポート(P):	COM27	OK
ボー・レート(B):	115200	
データ(D):	8 bit	キャンセル
パリティ(A):	none	ヘルプ(H)
ストップ(S):	1 bit	
フロー制御(F):	none	

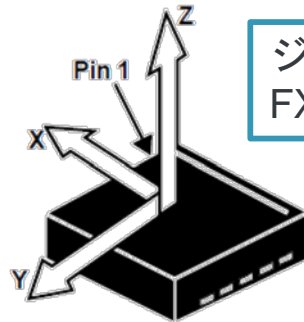
送信遅延

0	ミリ秒/字(C)	0	ミリ秒/行(L)
---	----------	---	----------

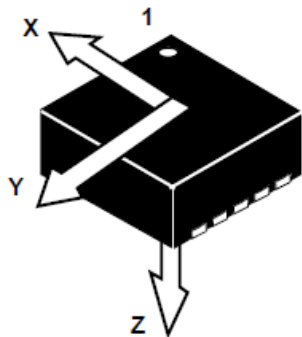
FRDM-FXS-MULTI、各センサの軸方向



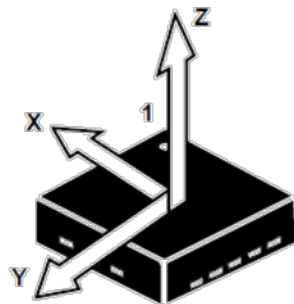
FXAS21000



MMA8652



MAG3110



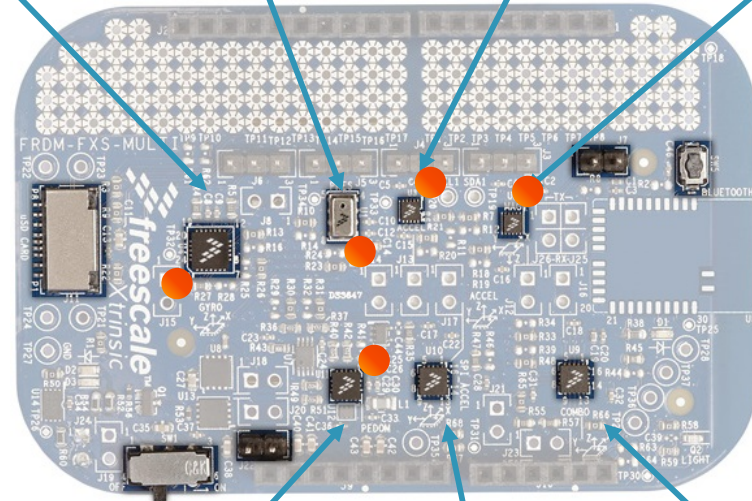
MMA9553

ジャイロ
FXAS21000

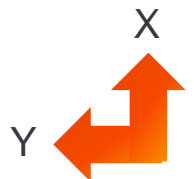
気圧センサ
MPL3115A2

加速度センサ
MMA8652

磁気センサ
MAG3110



● 1 pin



MCU+加速度
MMA9553

加速度センサ
FXLS8471Q

加速度+磁気
FXOS8700CQ

FRDM-FXS-MULTI、用意するサンプルコード概要

サンプルコードによるセンサの初期設定と戻り値

センサ	品名	初期設定	Data register値のdescription
気圧計	MPL3115A2	<ul style="list-style-type: none">気圧モードでに設定サンプリング=512msに設定気圧値は下2Bitが小数点温度は下4Bitが小数点	<ul style="list-style-type: none">気圧値[count].。 (単位変換は count / 4 [pa])温度[count] (単位変換は count / 16 [°C])
加速度	MMA8652	<ul style="list-style-type: none">±2gモードに設定分解能=1024 [g/LSB]サンプリング=200Hz	<ul style="list-style-type: none">X/Y/z軸の加速度値[count]. (単位変換は count / 1024 [g])
磁気	MAG3110	<ul style="list-style-type: none">サンプリング=20Hzに設定分解能=0.1 [uT/LSB]	<ul style="list-style-type: none">X/Y/z軸の磁気 [count]. (単位変換は count * 0.1 [uT])
ジャイロ	FXAS21000	<ul style="list-style-type: none">フルスケール1600dpsに設定分解能=0.2 [dps/LSB]サンプリング=200Hz	<ul style="list-style-type: none">X/Y/z軸の角加速度値[count]. (単位変換は count * 0.2 [dps])
歩数計	MMA9553	<ul style="list-style-type: none">歩数計として初期化身長175cm、体重80kg、女性と設定	<ul style="list-style-type: none">歩数[steps]歩行距離[m]速度[m/h]消費カロリーアクティビティ状態[4段階表示]Sleepカウント

FRDM-FXS-MULTI、I2C通信

I2C通信フォーマットとサンプルコード（MMA9553以外）

- 通信パケットの最初=I2Cアドレス
- 2番目はR/Wするレジスタのアドレス
- 3番目以降がR/Wするデータ

Single-byte Write : [IIC_RegWrite\(\)](#) 関数

Master	ST	I2Cアドレス+W		レジスタ・アドレス		Data[7:0]		SP
Slave			AK		AK		AK	

ST: Start Condition
SP: Stop Condition
AK: Acknowledge

Multiple-byte Write : [IIC_RegWriteN\(\)](#) 関数

Master	ST	I2Cアドレス+W		レジスタ・アドレス		Data[7:0]		Data[7:0]		...	Data[7:0]		SP
Slave			AK		AK		AK		AK	...		AK	

デバイス	I2Cアドレス(7bit)
MMA8652 (加速度)	0x1D << 1
MPL3115 (気圧)	0x60 << 1
MAG3110 (磁気)	0x0E << 1
FXAS21000 (ジャイロ)	0x20 << 1
MMA9553 (加速度+MCU)	0x4C << 1

I2Cアドレスは8Bit中のBit7~1で表わされる。Bit0でRead(1)/Write(0)を示す。例えばMMA8652への書き込み時のI2Cアドレスは0x3A。読み出し時は0x3Bと変わる。

FRDM-FXS-MULTI、I2C通信

I2C通信フォーマットとサンプルコード（MMA9553以外）

Single-byte Read : [IIC_RegRead\(\)](#) 関数

Master	ST	I2Cアドレス+W		レジスタ・アドレス		SR	I2Cアドレス+R			NAK	SP
Slave			AK		AK			AK	Data[7:0]		

Multiple-byte Read : [IIC_RegReadN\(\)](#) 関数

Master	ST	I2Cアドレス+W		レジスタ・アドレス		SR	I2Cアドレス+R					...		NAK	SP
Slave			AK		AK			AK	Data[7:0]	AK	Data[7:0]	AK	...	Data[7:0]	

ST: Start Condition
SP: Stop Condition
SR: Repeated Start Condition
AK: Acknowledge
NAK: No Acknowledge

I2Cの通信仕様の詳細はNXPのユーザマニュアルを参照してください。

http://www.nxp.com/documents/user_manual/UM10204_JA.pdf

FRDM-FXS-MULTI、MMA9553とのI2C通信

MMA9553のI2C通信フォーマットとサンプルコード

Multiple-byte Write : [MMA9553_SetConfig \(\)](#) 関数

					MB0		MB1		MB2	
Master	ST	I2Cアドレス+W		Mailbox(0x00)		APP_ID(0x15)		コマンド(0x20)		レジスタのオフセット
Slave			AK		AK		AK		AK	AK

	MB3		MB4		MB5			MB19		
	送信Byte数(16)		Data0[7:0]		Data1[7:0]		...	Data15[7:0]		SP
		AK		AK		AK	...		AK	

Data0~Data15=各レジスタ設定値

レジスタの詳細はMMA9553LSWRM.pdfのp18、2.3.2章を参照。

Multiple-byte Read: [MMA9553_DumpStatus\(\)](#) 関数

						MB0		MB1	
Master	ST	I2Cアドレス+W		Mailbox(0x00)	SR	I2Cアドレス+R		APP_ID(0x15)	COCO
Slave			AK		AK		AK		AK

	MB2		MB3		MB4		MB5			MB15		
									...		NAK	SP
	読み出しByte数	AK	読み出し要求Byte数	AK	Data0 [7:0]	AK	Data1 [7:0]	AK	...	Data11[7:0]		

Data0~Data11=各レジスタ読み出し値

レジスタの詳細はMMA9553LSWRM.pdfのp19、2.3.3章を参照。

各センサの紹介Webページ：

FRDM-FXS-MULTI

http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=FRDM-FXS-MULTI

FXAS21000

http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=FXAS21000

MPL3115A2

http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MPL3115A2

MMA8652

http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MMA8652FC

MGA3110

http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MAG3110

MMMA9553

http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MMA9553L



Softwareの実装準備

センサーを動作させるIIC driver とセンサー driver

PLM

Interface

1. IIC_Interface.h

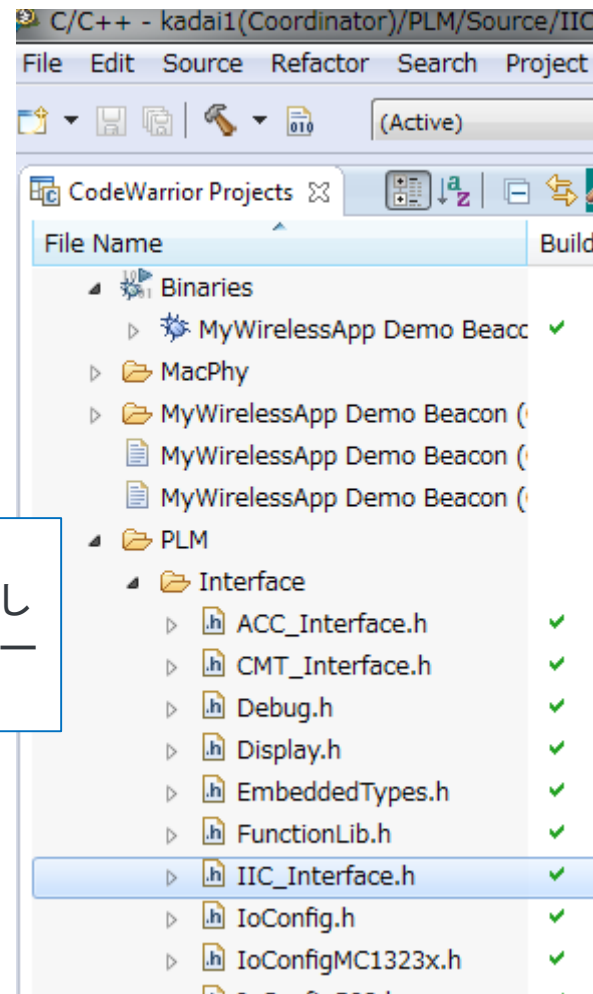
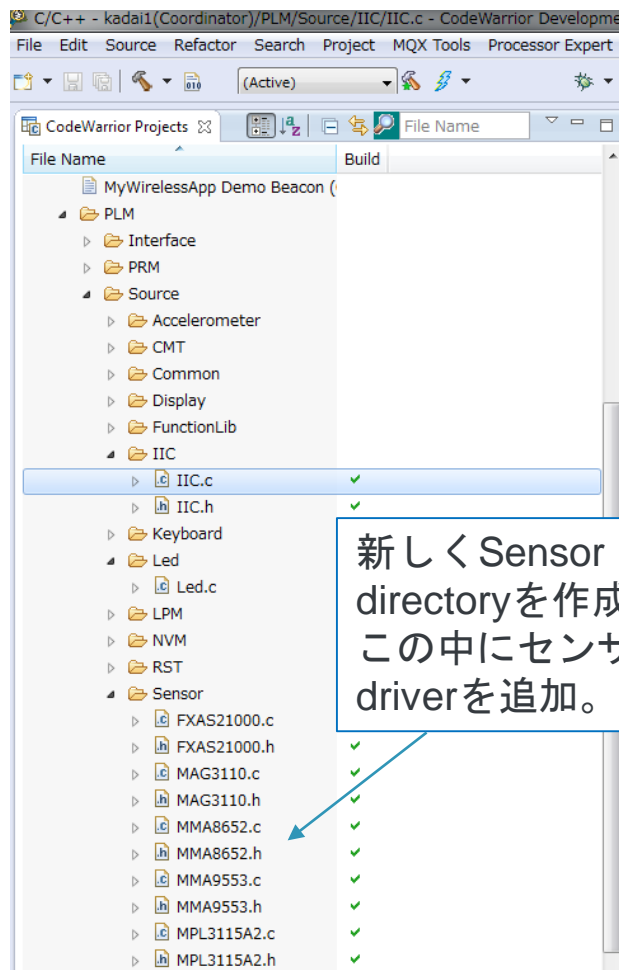
Source

IIC

1. IIC.c
2. IIC.h

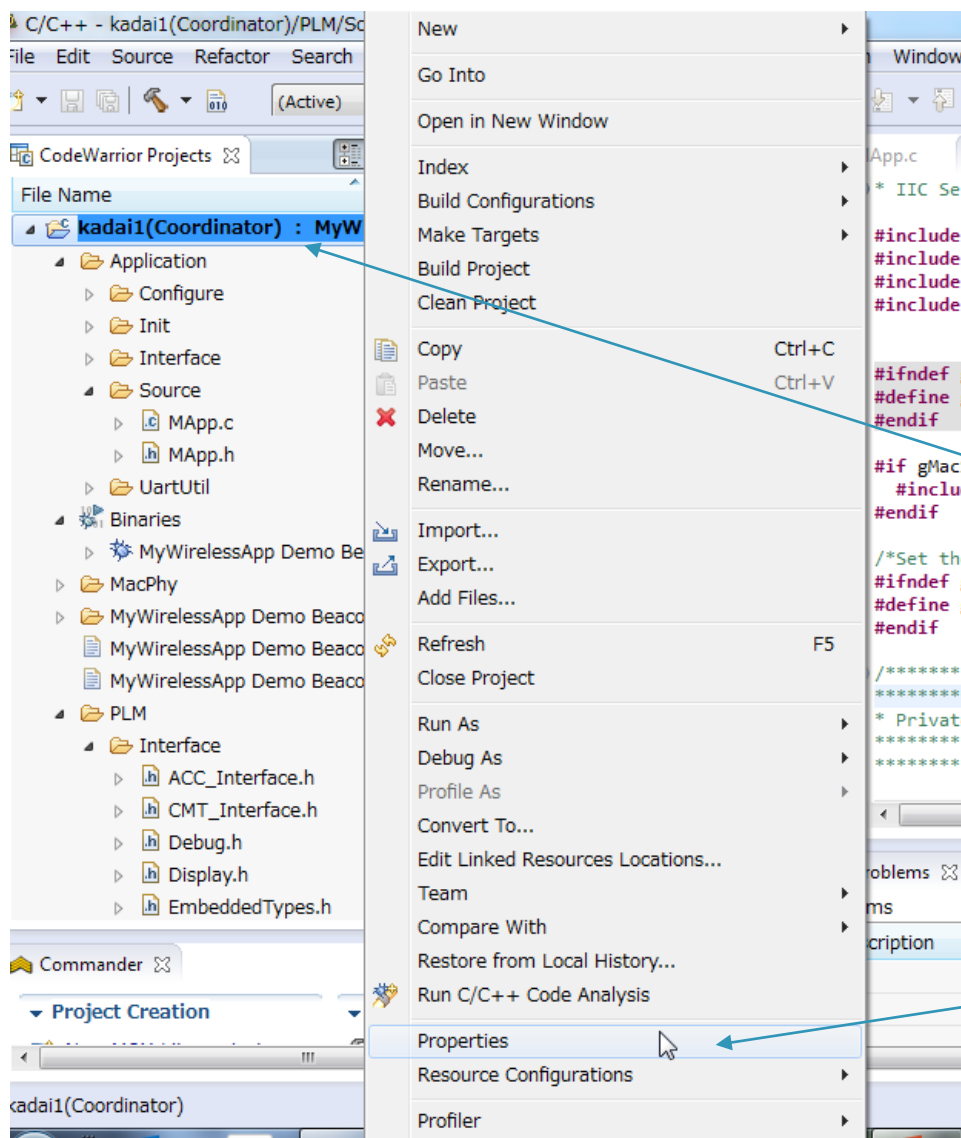
Sensor

1. FXAS21000.c
2. FXAS21000.h
3. MAG3110.c
4. MAG3110.h
5. MMA8652.c
6. MMA8652.h
7. MMA9553.c
8. MMA9553.h
9. MPL3115A2.c
10. MPL3115A2.h



* End-deviceのproject fileに実装する時は、左側に記載されているFilesを追加お願いします。IIC関係のfileは、全部入れ替えてください。

Include File Pathの追加 - 1

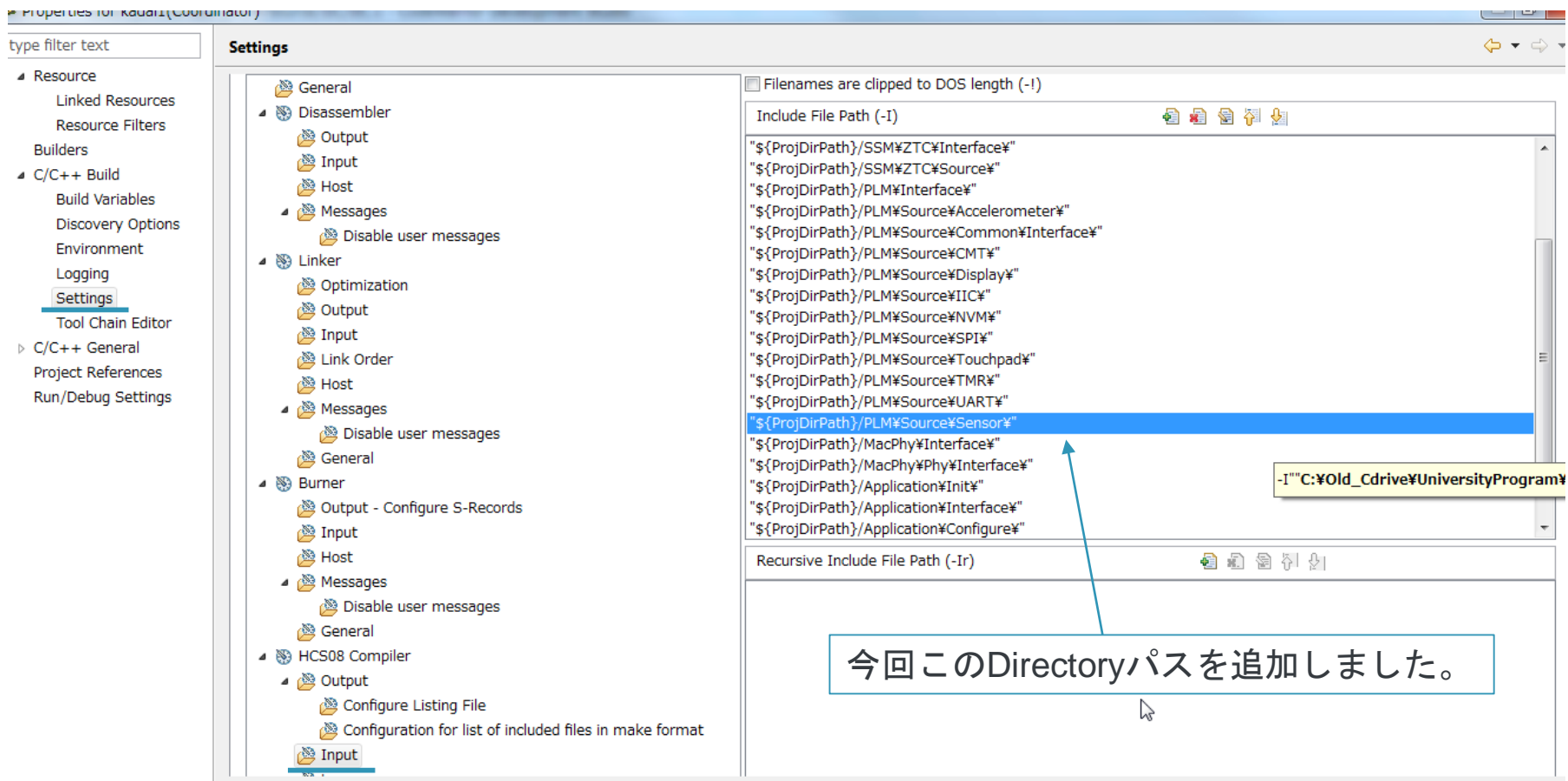


新しく Sensor directory を作成し
この中に センサー driver を追加したので
Include file path の追加設定

ここで一度クリックしてアクティブにし、
右クリックにて左側のメニューを参照

Properties をクリック

Include File Pathの追加 - 2



The screenshot shows the 'Settings' window for the 'kagaz(Coordinator)' project. The left sidebar lists various settings categories, with 'Settings' selected under 'C/C++ Build'. The main pane shows the 'Include File Path (-I)' list, which contains several paths. A blue arrow points from a text box to the path `"${ProjDirPath}/PLMSourceSensor"` in the list. A yellow tooltip shows the expanded path: `-I"C:\Old_Cdrive\UniversityProgram\`. Below the list, a text box contains the Japanese text: `今回このDirectoryパスを追加しました。`

type filter text

Settings

- Resource
 - Linked Resources
 - Resource Filters
- Builders
 - C/C++ Build
 - Build Variables
 - Discovery Options
 - Environment
 - Logging
 - Settings
 - Tool Chain Editor
 - C/C++ General
 - Project References
 - Run/Debug Settings

General

- Disassembler
 - Output
 - Input
 - Host
- Messages
 - Disable user messages
- Linker
 - Optimization
 - Output
 - Input
 - Link Order
 - Host
- Messages
 - Disable user messages
- General
- Burner
 - Output - Configure S-Records
 - Input
 - Host
- Messages
 - Disable user messages
- General
- HCS08 Compiler
 - Output
 - Configure Listing File
 - Configuration for list of included files in make format
 - Input

FileNames are clipped to DOS length (-I)

Include File Path (-I)

- "\${ProjDirPath}/SSM#ZTC#Interface"
- "\${ProjDirPath}/SSM#ZTC#Source"
- "\${ProjDirPath}/PLM#Interface"
- "\${ProjDirPath}/PLM#Source#Accelerometer"
- "\${ProjDirPath}/PLM#Source#Common#Interface"
- "\${ProjDirPath}/PLM#Source#CMT"
- "\${ProjDirPath}/PLM#Source#Display"
- "\${ProjDirPath}/PLM#Source#IIC"
- "\${ProjDirPath}/PLM#Source#NVM"
- "\${ProjDirPath}/PLM#Source#SPI"
- "\${ProjDirPath}/PLM#Source#Touchpad"
- "\${ProjDirPath}/PLM#Source#TMR"
- "\${ProjDirPath}/PLM#Source#UART"
- "\${ProjDirPath}/PLM#Source#Sensor"
- "\${ProjDirPath}/MacPhy#Interface"
- "\${ProjDirPath}/MacPhy#Phy#Interface"
- "\${ProjDirPath}/Application#Init"
- "\${ProjDirPath}/Application#Interface"
- "\${ProjDirPath}/Application#Configure"

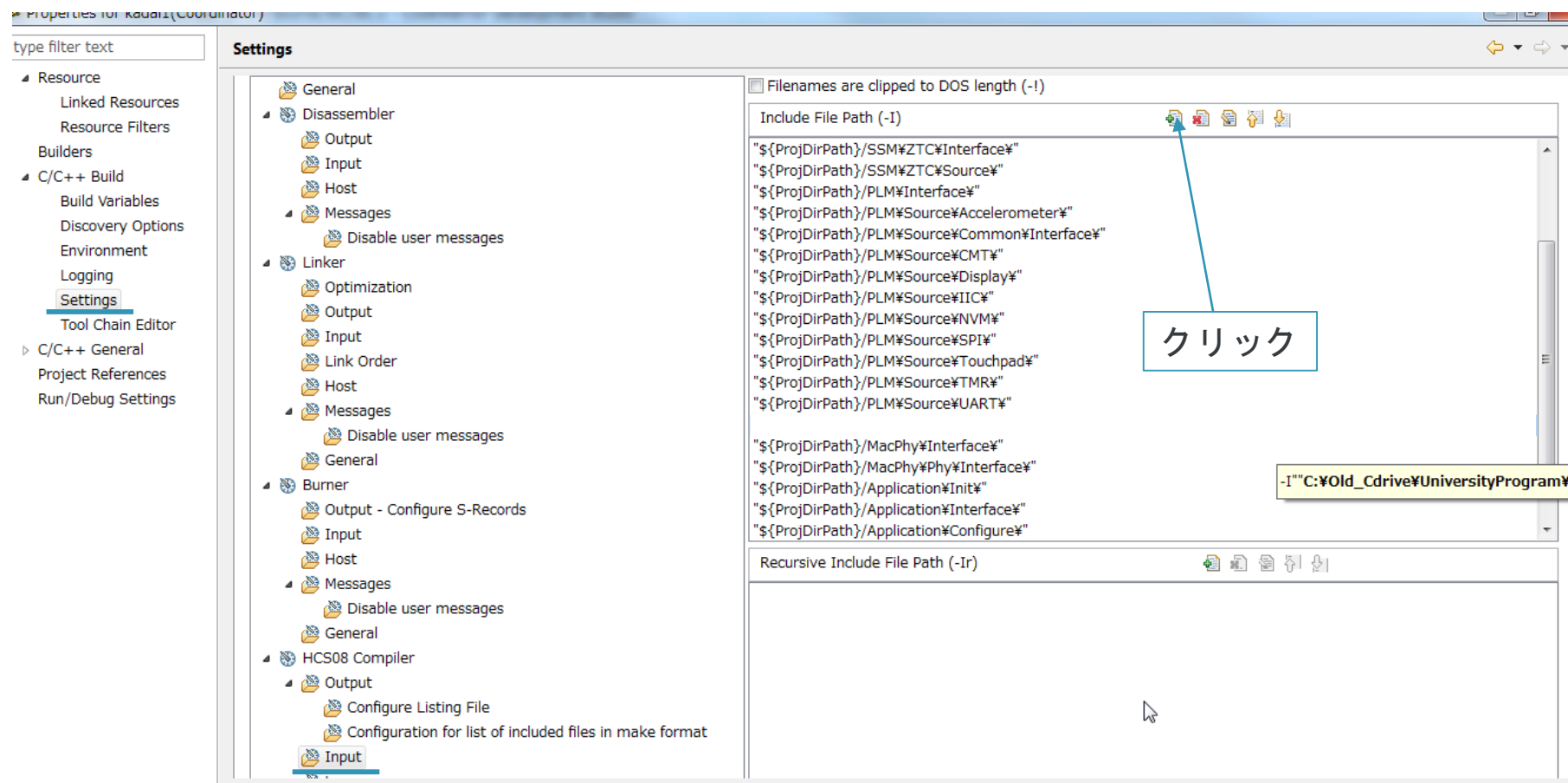
Recursive Include File Path (-Ir)

-I"C:\Old_Cdrive\UniversityProgram\

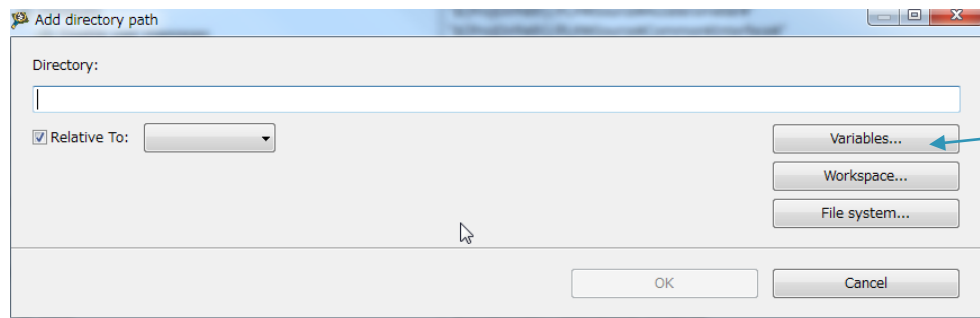
今回このDirectoryパスを追加しました。

Include File Pathの追加 - 3

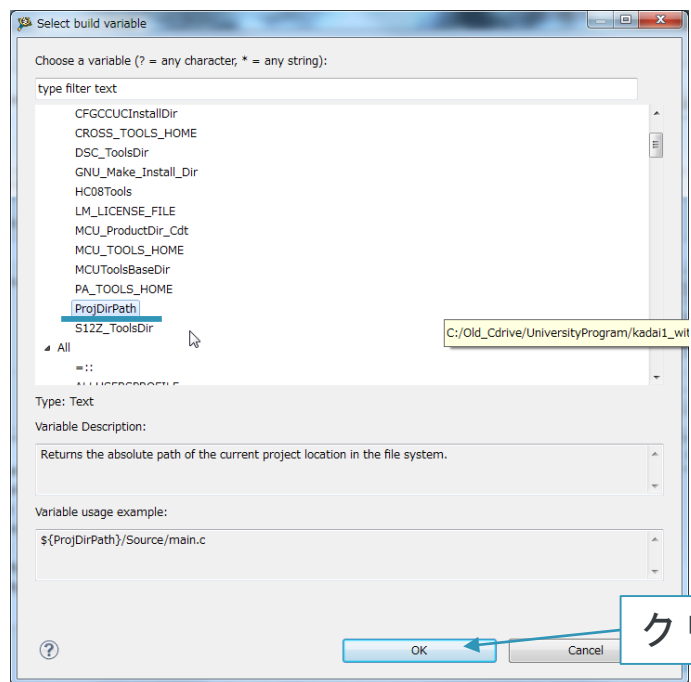
Include file pathの追加方法を実際に行う方法をご紹介します。



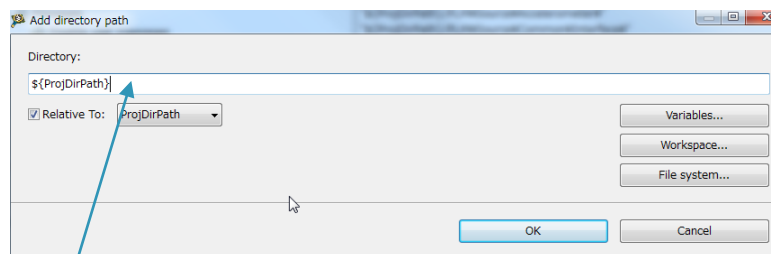
Include file pathの追加方法 - 4



クリック



クリック



残りのパスを以下のように入力してください。
\${ProjDirPath}/PLM¥Source¥Sensor¥



IIC driver実装

IIC_ModuleInitの変更

```
void IIC_ModuleInit(void)
{
    #if gIIC_Enabled_d
        /* Configure the I2C hardware peripheral */
        mIIC_C_c = mIICC_Reset_c;
        /* Clear the I2C Rx software buffer */
        mIICRxBufLeadingIndex = mIICRxBufTrailingIndex = mIICRxBufferByteCount = 0;
        pfIICSlaveRxCallBack = NULL;
        pfIICMasterTxCallBack = NULL;

        #if gIIC_Slave_TxDataAvailableSignal_Enable_c
            /* Configure as output the GPIO that will be used to signal to the host that
             the blackBox I2C slave device has data to be transmitted */
            /* Signal to the host that there are no data available to be read */
            gIIC_TxDataAvailablePortDataReg_c |= gIIC_TxDataAvailablePinMask_c;
            gIIC_TxDataAvailablePortDDirReg_c |= gIIC_TxDataAvailablePinMask_c;
        #endif

        mIIC_S_c = mIICS_Init_c;
        mIIC_F_c = gIIC_DefaultBaudRate_c;

        #if defined(PROCESSOR_MC1323X)
            IIC1C2 = mIICx2_Init_c;
        #endif
        /* Create I2C module main task */
        //IIC_TaskInit(); この関数をdisable

        mIIC_C_c = mIICC_IICEN_c; // Polling base
    #endif
}
```

Baudrateは、100kHz

```
/* Default baud rate. */
#ifndef gIIC_DefaultBaudRate_c
//Ori #define gIIC_DefaultBaudRate_c gIIC_BaudRate_100000_c
#define gIIC_DefaultBaudRate_c gIIC_BaudRate_100000_c
#endif
```

Defaultのままだと割り込み実装になっているので
Polling baseに変更

Multi write 関数

IIC Slave address

Register address

Bytes

Write data

```
void IIC_RegWriteN(uint8_t address, uint8_t reg1, uint8_t N, uint8_t *array)
{
    mIIC_C_c |= mIICC_TX_c;           // Transmit Mode
    IIC_Start();                     // Send Start
    IIC_CycleWrite(address);          // Send IIC "Write" Address
    IIC_CycleWrite(reg1);             // Send Register
    while (N>0)                      // Send N Values
    {
        IIC_CycleWrite(*array);
        array++;
        N--;
    }
    IIC_Stop();                      // Send Stop
}
```

Multi read 関数

IIC Slave address

Register address

Bytes

Read data

```
void IIC_RegReadN(uint8_t address, uint8_t reg1, uint8_t N, uint8_t *array)
{
    uint8_t read_data;

    mIIC_C_c |= mIICC_TX_c;           // Transmit Mode
    IIC_Start();                     // Send Start
    IIC_CycleWrite(address);         // Send IIC "Write" Address
    IIC_CycleWrite(reg1);            // Send Register
    IIC_RepeatStart();               // Send Repeat Start
    IIC_CycleWrite(address+1);       // Send IIC "Read" Address
    read_data = IIC_CycleRead(N);    // *** Dummy read: reads "IIC_ReadAddress" value ***
    while (N>1)                     // Read N-1 Register Values
    {
        N--;
        read_data = IIC_CycleRead(N);
        *array = read_data;
        array++;
    }
    read_data = IIC_StopRead();
    *array = read_data;             // Read Last value
}
```

Single write & read 関数

IIC Slave address

Register address

Write data

```
void IIC_RegWrite(uint8_t address, uint8_t reg, uint8_t val)
{
    mIIC_C_c |= mIICC_TX_c;           // Transmit Mode
    IIC_Start();                       // Send Start
    IIC_CycleWrite(address);           // Send IIC "Write" Address
    IIC_CycleWrite(reg);               // Send Register
    IIC_CycleWrite(val);               // Send Value
    IIC_Stop();                        // Send Stop
}
```

Read data

IIC Slave address

Register address

```
uint8_t IIC_RegRead(uint8_t address, uint8_t reg)
{
    volatile uint8_t b;

    mIIC_C_c |= mIICC_TX_c;           // Transmit Mode
    IIC_Start();                       // Send Start
    IIC_CycleWrite(address);           // Send IIC "Write" Address
    IIC_CycleWrite(reg);               // Send Register
    IIC_RepeatStart();                 // Send Repeat Start
    IIC_CycleWrite(address+1);         // Send IIC "Read" Address
    b = IIC_CycleRead(1);               // *** Dummy read: reads "IIC_ReadAddress" value
    b = IIC_StopRead();                // Send Stop Read command
    return b;
}
```

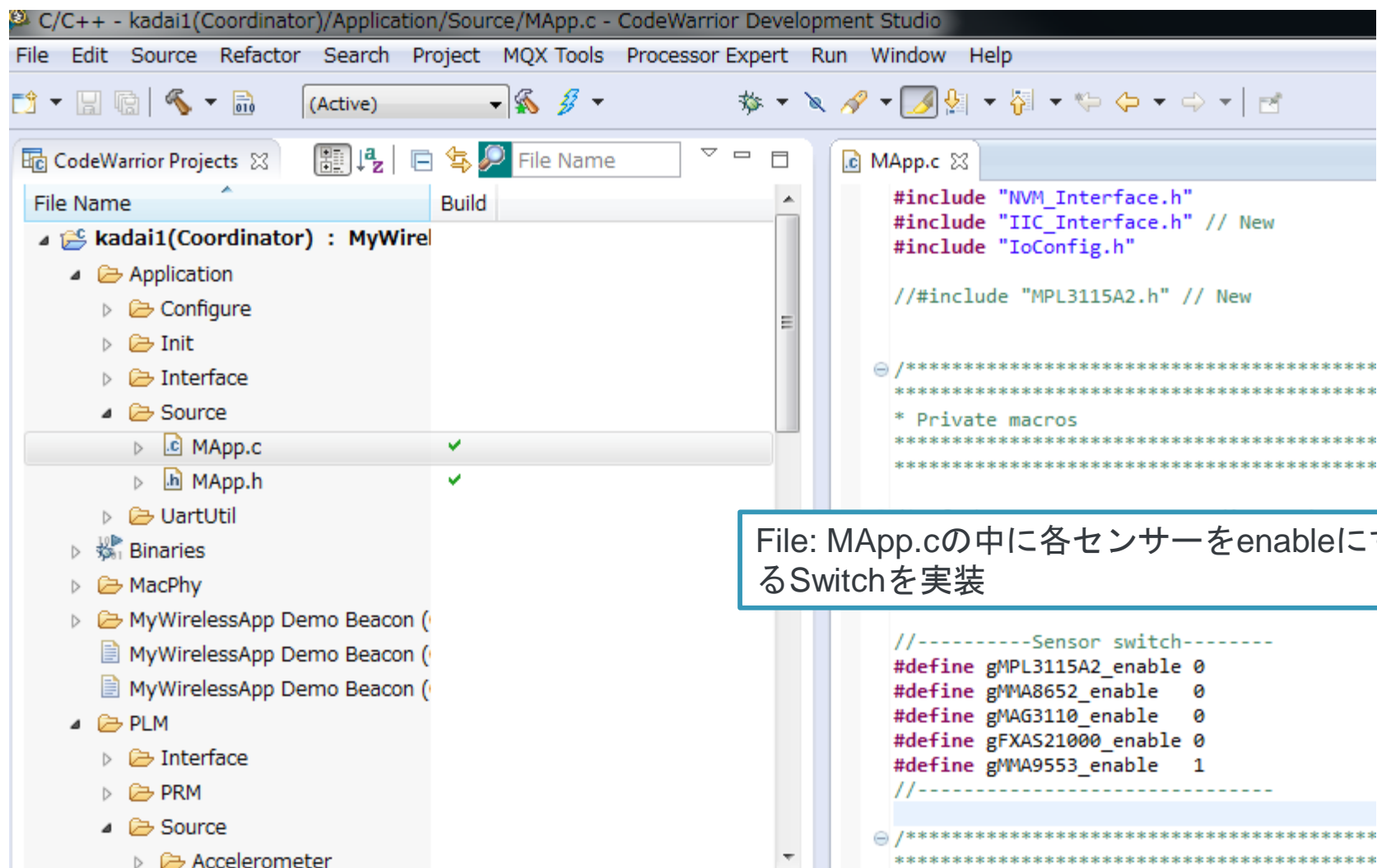
部品関数一覧

1. void IIC_Start(void)
2. void IIC_CycleWrite(uint8_t bout)
3. void IIC_Stop(void)
4. void IIC_RepeatStart(void)
5. uint8_t IIC_CycleRead(uint8_t byteLeft)
6. uint8_t IIC_StopRead(void)



各センサーのconfigurationの実装

各センサーをenableにするswitch



The screenshot shows the CodeWarrior Development Studio interface. On the left, the 'CodeWarrior Projects' pane displays the project structure for 'kadai1(Coordinator) : MyWirelessApp'. The 'Source' folder is expanded, showing 'MApp.c' and 'MApp.h'. On the right, the 'MApp.c' file is open in the editor. The code includes headers for 'NVM_Interface.h', 'IIC_Interface.h', and 'IoConfig.h'. It also contains a section for 'Private macros' and a 'Sensor switch' section with several macros defined to enable or disable sensors.

```
#include "NVM_Interface.h"
#include "IIC_Interface.h" // New
#include "IoConfig.h"

// #include "MPL3115A2.h" // New

/*****
*****
***** Private macros
*****
*****
*****/

//-----Sensor switch-----
#define gMPL3115A2_enable 0
#define gMMA8652_enable 0
#define gMAG3110_enable 0
#define gFXAS21000_enable 0
#define gMMA9553_enable 1
//-----

/*****
*****
*****/
```

File: MApp.cの中に各センサーをenableにするSwitchを実装

IICのconfigurationとIIC Bus resetの追加

```
MApp.c  IIC.c  FXAS21000.c  IIC_Interface.h

void MApp_init(void)
{
    /* The initial application state */
    gState = stateInit;
    /* Reset number of pending packets */
    mcPendingPackets = 0;

    /* Initialize the MAC 802.15.4 extended address */
    Init_MacExtendedAddress();
    /* register keyboard callback function */
    KBD_Init(App_HandleKeys);
    /* Initialize SPI Module */
    SPI_Init();

    /* initialize LCD Module */
    LCD_Init();
    /* initialize LED Module */
    LED_Init();
    /* Initialize the LPM module */
    PWRLib_Init();
    /* Initialize the UART so that we can print out status messages */
    UartX_SetBaud(gUartDefaultBaud_c);
    UartX_SetRxCallBack(UartRxCallBack);

    /* Initialize the IIC module */
    IIC_Bus_Reset(); // Reset IIC bus to prevent freezing sensor module using GF
    IIC_ModuleInit(); // New
```

各センサーのenable及び初期化

```
MApp.c IIC.c FXAS21000.c IIC_Interface.h

LED_Init();
/* Initialize the LPM module */
PWRLib_Init();
/* Initialize the UART so that we can print out status mess
UartX_SetBaud(gUartDefaultBaud_c);
UartX_SetRxCallBack(UartRxCallBack);

/* Initialize the IIC module */
IIC_Bus_Reset(); // Reset IIC bus to prevent freezing sens
IIC_ModuleInit(); // New

#if gMPL3115A2_enable
    MPL3115A2_Init();
#endif

#if gMMA8652_enable
    MMA8652_Init();
#endif

#if gMAG3110_enable
    MAG3110_Init();
#endif

#if gFXAS21000_enable
    FXAS21000_Init();
#endif

#if gMMA9553_enable
    MMA9553_Init();
#endif

/* initialize buzzer (NCB, SRB only) */
BuzzerInit();
```

```
MApp.c IIC.c FXAS21000.c

// #include "MPL3115A2.h" // New

/* Private macros
*****
* Private macros
*****

/* If there are too many pending pack
/* receive mMaxKeysToReceive_c chars.
/* The chars will be send over the ai
#define mMaxKeysToReceive_c 32

//-----Sensor switch-----
#define gMPL3115A2_enable 0
#define gMMA8652_enable 0
#define gMAG3110_enable 0
#define gFXAS21000_enable 0
#define gMMA9553_enable 1
//-----
```

このケースでは、歩数計のみがactive

各センサーのRegister値の表示部分

File: Mapp.c 関数 void AppTask(event_t events)

```
case stateListen:
    /* Stay in this state forever.
       Transmit the data received on UART */
    if (events & gAppEvtMessageFromMLME_c)
    {
        /* Get the message from MLME */
        if (pMsgIn)
        {
            /* Process it */
            ret = App_HandleMlmeInput(pMsgIn);
            /* Messages from the MLME must always be freed. */
        }
    }

    if (events & gAppEvtRxFromUart_c)
    {
        /* get byte from UART */
        App_TransmitUartData();
    }

    #if gMPL3115A2_enable
        MPL3115A2_dump(events);
    #endif

    #if gMMA8652_enable
        MMA8652_dump(events);
    #endif

    #if gMAG3110_enable
        MAG3110_dump(events);
    #endif

    #if gFXAS21000_enable
        FXAS21000_dump(events);
    #endif

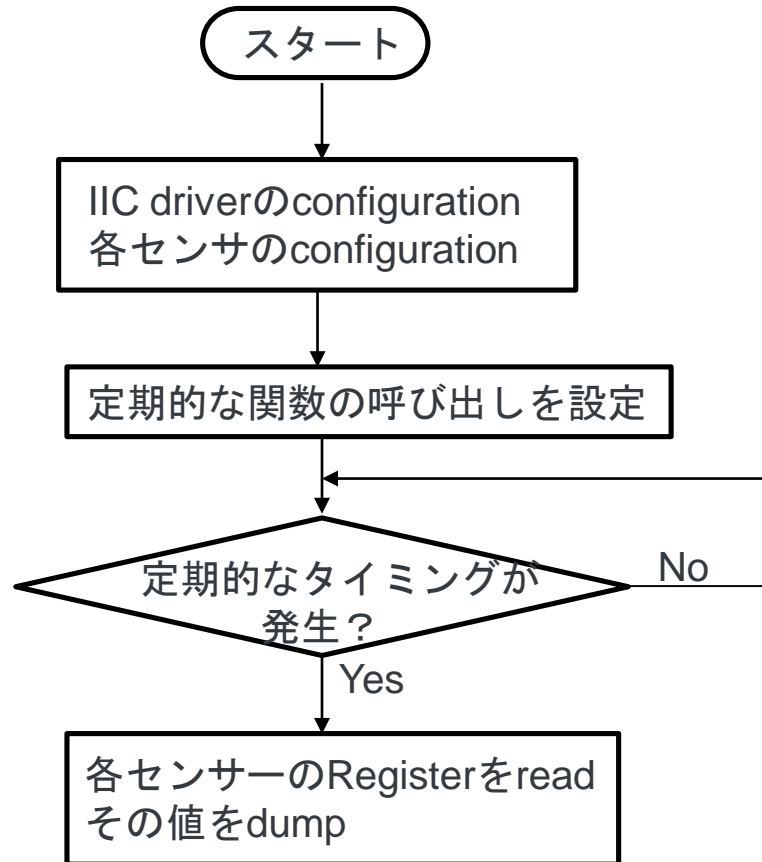
    #if gMMA9553_enable
        MMA9553_dump(events);
    #endif
```

定期的に以下の処理を実施

1. センサデータを読む
2. データ値をTerminal softへdump



センサーデータ値のread timing



```
// Start to receive periodical data
if(MMA9553_start_flag){
    MMA9553_start_flag = 0;
    MMA9553_Start_Periodical_data();
}
```

```
tmp = (uint16_t)MMA9553_CatchSensorData(i);
UartUtil_PrintHex((uint8_t *)&tmp, 2, 1);
```

現在のポーリング周期設定

センサー群	現在のポーリング周期
FXAS2100: 3軸ジャイロセンサ	100ms
MMA9553: インテリジェント歩数計 (MCU+加速度)	1000ms
MPL3115A2: 気圧センサ	700ms
MMA8652: 3軸加速度センサ	700ms
MAG3110: 3軸磁気センサ	700ms

周期は、変更可能

FXAS2100: 3軸ジャイロセンサは、角速度を求めるため周期が短くなる。よって他のセンサと同時にdisplayするのは、避けてください。

同時に複数センサのregister値をdisplayしたケース

```
Tera Term - [未接続] VT
ファイル(F) 編集(E) 設定(S) コントロール(Q) ウィンドウ(W) ヘルプ(H)

-----
MAG3110(16bit)
X axis : 0xFB79 *0.1 [uT]
Y axis : 0x04D9 *0.1 [uT]
Z axis : 0x0084 *0.1 [uT]
-----

MMA9553
X Act_data : 0x0001
X Step_data : 0x0000
X Dist_data : 0x0000
X Speed_data : 0x0000
X Cal_data : 0x0000
X Sleep_cnt_data : 0x007F
-----

MPL3115A2(P: Int18,Float2, T: Int8, Float4)
Press : 0x00063642 /4 [Pa]
Temp : 0x0000017E /16 [C]
-----

MMA8652(2g: Int2,Float10)
X axis : 0x001A /1024 [g]
Y axis : 0xFFFF /1024 [g]
Z axis : 0x0426 /1024 [g]
-----

MAG3110(16bit)
X axis : 0xFB7A *0.1 [uT]
Y axis : 0x04D7 *0.1 [uT]
Z axis : 0x0086 *0.1 [uT]
-----

//-----Sensor switch-----
#define gMPL3115A2_enable 1
#define gMMA8652_enable 1
#define gMAG3110_enable 1
#define gFXAS21000_enable 0
#define gMMA9553_enable 1
//-----
```

符号付きの値で表示されている

Mapp.hへの追加

```
#define gAppEvtDummyEvent_c      (1 << 0)
#define gAppEvtRxFromUart_c      (1 << 1)
#define gAppEvtMessageFromMLME_c (1 << 2)
#define gAppEvtMessageFromMCPS_c (1 << 3)
#define gAppEvtStartCoordinator_c (1 << 4)

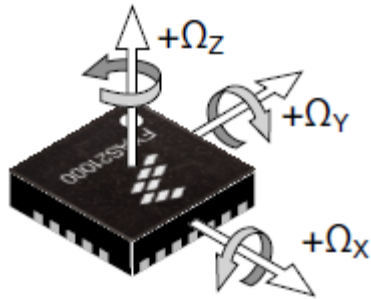
#define gAppEvt_FromMPL3115A2_c  (1 << 7) // New
#define gAppEvt_FromMMA8652_c    (1 << 9) // New
#define gAppEvt_FromMAG3110_c    (1 << 10) // New
#define gAppEvt_FromFXAS21000_c  (1 << 11) // New
#define gAppEvt_FromMMA9553_c    (1 << 12) // New
```

AppTaskの中で使用する
各センサー用のeventを定義



FXAS2100: 3軸ジャイロセンサ

FXAS2100: 3軸ジャイロセンサ



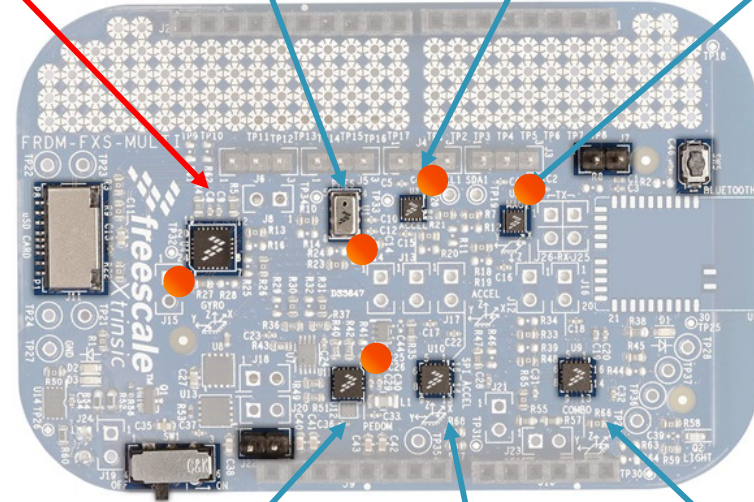
FXAS21000

ジャイロ
FXAS21000

気圧センサ
MPL3115A2

加速度センサ
MMA8652

磁気センサ
MAG3110



1 pin

X

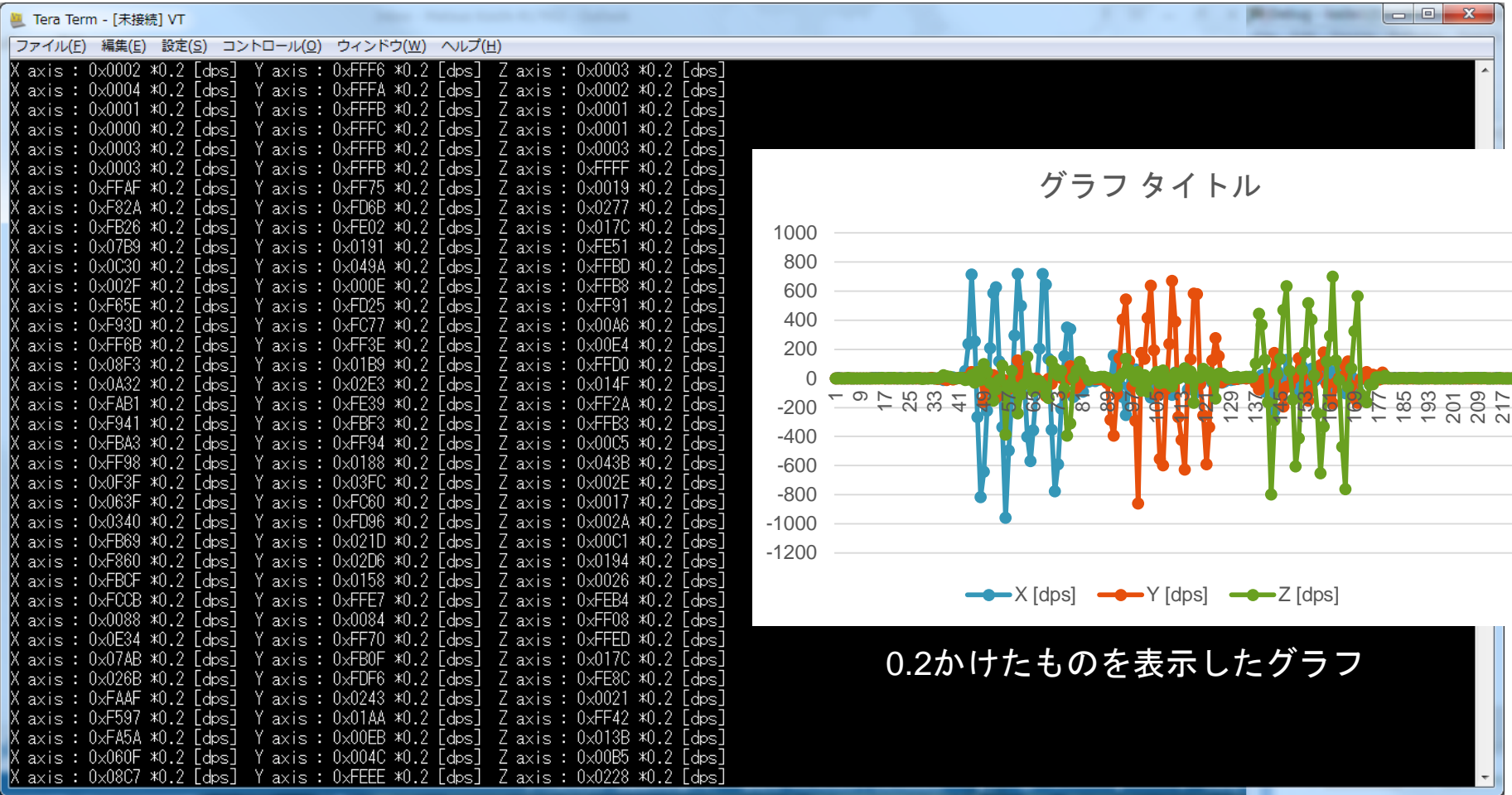
Y

MCU+加速度
MMA9553

加速度センサ
FXLS8471Q

加速度+磁気
FXOS8700CQ

FXAS2100: 3軸ジャイロセンサの表示



使用関数

```
1. void FXAS21000_Init(void);
    1. void FXAS21000_SetCallBack(void
        (*pfCallBack)(void));
    2. void (*pfFXAS21000_CallBack)(void);
    3. void FXAS21000_int(void);
2. void FXAS21000_dump(event_t events);
    1. void FXAS21000_Start_Periodical_data(void);
    2. void FXAS21000_Periodical_data(uint8_t timerId);
    3. int16_t FXAS21000_CatchSensorData(uint8_t
        number);
    4. void FXAS21000_CallBack(void);
```

FXAS2100の初期化

```
void FXAS21000_Init(void){  
  ① FXAS21000_SetCallBack(FXAS21000_CallBack);  
  ② FXAS21000_int();  
  ③ FXAS21000_start_flag = 1;  
}
```

① call back関数の中でapplication task **AppTask**を call 設定

②

```
void FXAS21000_int(void){  
  IIC_RegWrite(FXAS21000_SlaveAddressIIC,0x13,0x00); // CTRL_REG1: Standby mode  
  IIC_RegWrite(FXAS21000_SlaveAddressIIC,0x0D,0x00); // CTRL_REG0: HPF=Off, 1600dps,  
  IIC_RegWrite(FXAS21000_SlaveAddressIIC,0x14,0x00); // CTRL_REG2: Interrupt=Off  
  IIC_RegWrite(FXAS21000_SlaveAddressIIC,0x13,0x02); // CTRL_REG1: ODR=200Hz, Active  
  mFXAS21000TimerID = TMR_AllocateTimer();  
}
```

定期的にcallする関数を設定するために専用のTimerを allocate.

③ **stateListen** stateになった後, FXAS2100を定期的呼び出すTimer関数をactiveにするflagをセット。

定期的なcallと表示

File: Mapp.cの中の

void AppTask(event_t events)

case *stateListen*:の中で以下の関数が定期的にcallされる。

```
#if gFXAS21000_enable
    FXAS21000_dump(events);
#endif
```

void FXAS21000_dump(event_t events)

```
void FXAS21000_dump(event_t events){  
  
volatile int16_t tmp;  
uint8_t i;  
  
// Start to receive periodical data  
if(FXAS21000_start_flag){  
    FXAS21000_start_flag = 0;  
    FXAS21000_Start_Periodical_data();  
}
```

```
void FXAS21000_Start_Periodical_data(void){  
    TMR_StartIntervalTimer(mFXAS21000TimerID, mFXAS21000Interval_c, FXAS21000_Periodical_data);  
}
```

=100ms

100ms定期毎にFXAS21000_Periodical_data関数がcallされる。
FXAS21000_Periodical_dataは、次頁に記載。

```
if (events & gAppEvt_FromFXAS21000_c)  
{  
    for(i=1; i<4; i++){  
        switch(i){  
            case 1:  
                UartUtil_Print("\n\rX axis : 0x", gAllowToBlock_d);  
                break;  
            case 2:  
                UartUtil_Print(" *0.2 [dps] Y axis : 0x", gAllowToBlock_d);  
                break;  
            case 3:  
                UartUtil_Print(" *0.2 [dps] Z axis : 0x", gAllowToBlock_d);  
                break;  
        }  
    }  
  
    tmp = (uint16_t)FXAS21000_CatchSensorData(i);  
    UartUtil_PrintHex((uint8_t *)&tmp, 2, 1);  
  
    if(i==3){  
        UartUtil_Print(" *0.2 [dps]", gAllowToBlock_d);  
    }  
}  
}
```

X,Y,Z[count]の各測定値を表示。
[count]*0.2=[dps]
*dps=degree par second (角加速度)

```
int16_t FXAS21000_CatchSensorData(uint8_t number){  
  
    int16_t catch_data;  
  
    switch(number){  
        case 1:  
            catch_data = mDataFrom_FXAS21000.xOutReg;  
            break;  
        case 2:  
            catch_data = mDataFrom_FXAS21000.yOutReg;  
            break;  
        case 3:  
            catch_data = mDataFrom_FXAS21000.zOutReg;  
            break;  
        default:  
            break;  
    }  
  
    return catch_data;  
}
```

void FXAS21000_Periodical_data(uint8_t timerId)

```
void FXAS21000_Periodical_data(uint8_t timerId){
```

```
    uint8_t rxData[7];
```

```
    volatile int16_t Read_data_16bit;
```

```
    (void) timerId; /* prevent compiler warning */
```

```
    rxData[0] = IIC_RegRead( FXAS21000_SlaveAddressIIC,0x00 ); // checking a STATUS-reg
```

```
    if( rxData[0] & 0x08 ){
```

```
        IIC_RegReadN( FXAS21000_SlaveAddressIIC, 0x01, 0x06, &rxData[1] ); // Read data from $0x01 to 0x06
```

```
        pfFXAS21000_CallBack(); // Set event in order to notify application in callback function.
```

```
        = TS_SendEvent(gAppTaskID_c, gAppEvt_FromFXAS21000_c)が実行されAppTaskを call される。
```

```
        Read_data_16bit = (int16_t)( rxData[1]<<8 );
```

```
        mDataFrom_FXAS21000.xOutReg = ( Read_data_16bit + (int16_t)rxData[2] )>>2;
```

```
        Read_data_16bit = (int16_t)(rxData[3]<<8);
```

```
        mDataFrom_FXAS21000.yOutReg = ( Read_data_16bit + (int16_t)rxData[4] )>>2;
```

```
        Read_data_16bit = (int16_t)(rxData[5]<<8);
```

```
        mDataFrom_FXAS21000.zOutReg = ( Read_data_16bit + (int16_t)rxData[6] )>>2;
```

```
    }
```

```
}
```

Buffer	Content
rxData[1]	X軸[13:6]
rxData[2]	X軸[5:0]
rxData[3]	Y軸[13:6]
rxData[4]	Y軸[5:0]
rxData[5]	Z軸[13:6]
rxData[6]	Z軸[5:0]

パラメータ	Description
mDataFrom_FXAS21000.xOutReg	X軸[14bit]
mDataFrom_FXAS21000.yOutReg	Y軸[14bit]
mDataFrom_FXAS21000.zOutReg	Z軸[14bit]

Call back関数の設定

センサconfigurationの中で設定

```
FXAS21000_SetCallBack(FXAS21000_CallBack);
```

```
void FXAS21000_SetCallBack(void(*pfCallBack)(void)){  
    pfFXAS21000_CallBack = pfCallBack;  
}
```

```
void FXAS21000_CallBack(void){  
    TS_SendEvent(gAppTaskID_c, gAppEvt_FromFXAS21000_c);  
    =AppTaskをcallします。 上記Eventをセットします。  
}
```

センサデータを
定期的にread
する関数

```
void FXAS21000_Periodical_data(uint8_t timerId){  
    pfFXAS21000_CallBack(); // Set event in order to notify application in callback function.
```

FXAS21000_CallBackを実行

以下の内容を実行

```
void AppTask(event_t events)  
{  
  
    void FXAS21000_dump(event_t events){  
        if (events & gAppEvt_FromFXAS21000_c) {
```




MMA9553: インテリジェント歩数計(MCU+加速度)

MMA9553の初期設定 (1/2)

2Byteレジスタ構成だが、Byte単位でR/W可能。

Table 2-4. Configuration registers

Offset address	Register	Access	Reset	Details
0x0	Sleep Minimum register	R/W	0x0000	"Sleep Minimum register" on page 18
0x2	Sleep Maximum register	R/W	0x0000	"Sleep Maximum register" on page 19
0x4	Sleep Count Threshold register	R/W	0x0001	"Sleep Count Threshold register" on page 19
0x6	Config/Step Length register	R/W	0x0000	"Configuration/Step Length register" on page 20
0x8	Height/Weight register	R/W	0xAF50	"Height/Weight register" on page 21
0xA	Filter register	R/W	0x0403	"Filter register" on page 21
0xC	Speed Period register	R/W	0x0501	"Speed Period/Step Coalesce register" on page 22
0xE	Activity Count Threshold register	R/W	0x0000	"Activity Count Threshold register" on page 22

MMA9553の初期設定 (2/2)

Offset	Register		設定値	詳細
0x0	Sleep Minimum Hi	MB4	0x0E	Sleepを認識する下限ベクトル・マグネチュード[0.244/LSB]
0x1	Lo	MB5	0x74	0x0E74(3700)*0.244= 903[mg]
0x2	Sleep Maximum Hi	MB6	0x11	Sleepを認識する上限ベクトル・マグネチュード[0.244/LSB]
0x3	Lo	MB7	0x94	0x1194(4500)*0.244=1098
0x4	Sleep Count Threshold Hi	MB8	0x32	Sleepを認識するカウント数
0x5	Lo	MB9	0x0A	0x0032=50[sample]
0x6	Config/Step Length Hi	MB10	0x80	再初期化する設定
0x7	Lo	MB11	0x00	
0x8	Height/Weight Hi	MB12	0xAF	身長[cm]。 0xAF=175cm。
0x9	Lo	MB13	0x50	体重[kg]。 0x50=80kg。
0xA	Filter register Hi	MB14	0x04	フィルタするステップ数。(誤カウント予防に3sec間に4ステップまでフィルタ)
0xB	Lo	MB15	0x03	性別、フィルタする時間[sec]。 女性、3sec。
0xC	Speed Period Hi	MB16	0x05	速度計算する時間[sec]。 5secに設定。有効範囲は0x02:0x05.
0xD	Lo	MB17	0x10	カロリー計算する時間[steps]。 16ステップに設定。
0xE	Activity Count Threshold Hi	MB18	0x00	Activityを認識する閾値[count]。 10countに設定。
0xF	Lo	MB19	0x0A	

各レジスタの詳細はMMA9553LSWRM.pdfのp20、2.3.4章を参照。

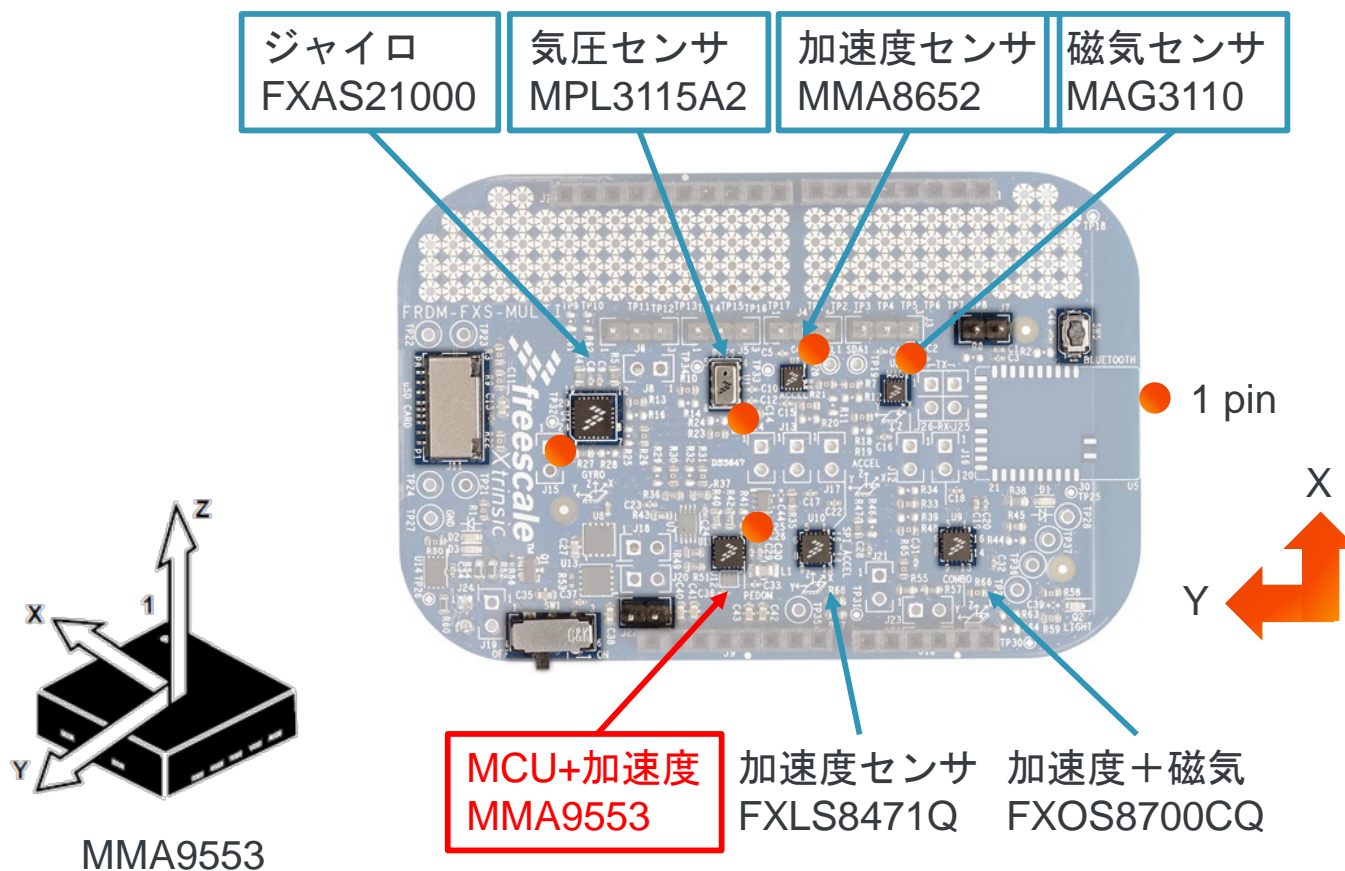
MMA9553のステータス・リード

2Byteレジスタ構成だが、Byte単位でR/W可能。

Offset	Register	MB	詳細
0x0	Status register	MB 4,MB5	ACTIVITYの状態 (Rest/Walking/Jogging/Running)
0x2	Step count register	MB 6,MB7	歩数
0x4	Distance register	MB 8,MB9	距離 [m]
0x6	Speed register	MB10,MB11	速度 [m/h]
0x8	Calories register	MB12,MB13	カロリー
0xA	Sleep Count register	MB14,MB15	Sleepカウント[回]

各レジスタの詳細はMMA9553LSWRM.pdfのp25、2.3.5章を参照。

MMA9553: インテリジェント歩数計(MCU+加速度)



MMA9553: インテリジェント歩数計(MCU+加速度)の表示

COM27:115200baud - Tera Term VT

ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

```
X Step_data : 0x0043
X Dist_data : 0x0037
X Speed_data : 0x1974
X Cal_data : 0x0003
X Sleep_cnt_data : 0x0000
```

-----J-----

MMA9553

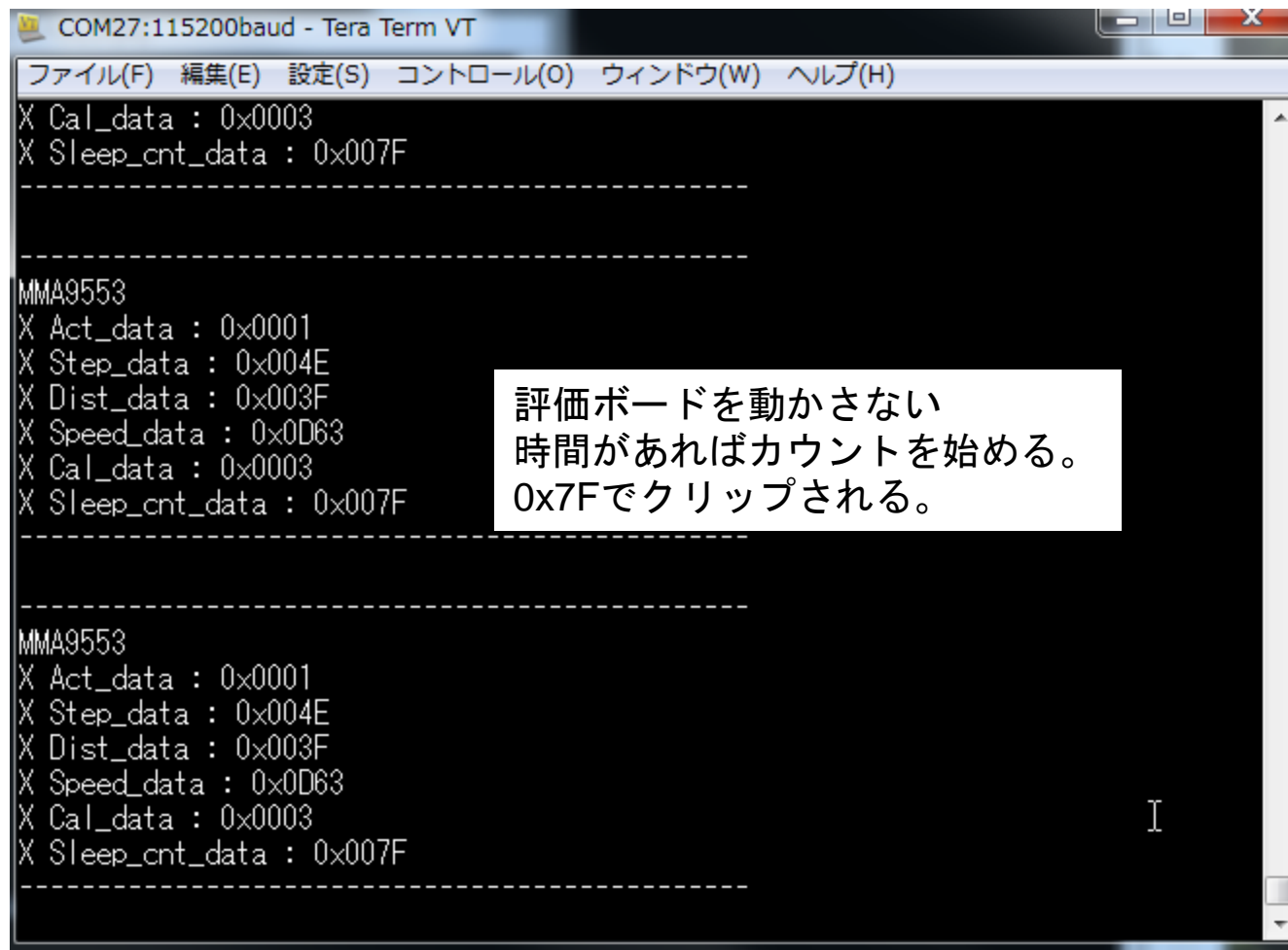
```
X Act_data : 0x0003
X Step_data : 0x0046
X Dist_data : 0x0039
X Speed_data : 0x19F5
X Cal_data : 0x0003
X Sleep_cnt_data : 0x0000
```

MMA9553

```
X Act_data : 0x0003
X Step_data : 0x0048
X Dist_data : 0x003B
X Speed_data : 0x19F5
X Cal_data : 0x0003
```

Description
アクティビティ(4 段階)[3bit]
歩数[16bit]] [steps]
距離[16bit]] [m]
速度[16bit][m/h]
消費カロリー[16bit]
Sleepした回数[16bit][count]

Sleepした回数



```
COM27:115200baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
X Cal_data : 0x0003
X Sleep_cnt_data : 0x007F
-----
MMA9553
X Act_data : 0x0001
X Step_data : 0x004E
X Dist_data : 0x003F
X Speed_data : 0x0D63
X Cal_data : 0x0003
X Sleep_cnt_data : 0x007F
-----
MMA9553
X Act_data : 0x0001
X Step_data : 0x004E
X Dist_data : 0x003F
X Speed_data : 0x0D63
X Cal_data : 0x0003
X Sleep_cnt_data : 0x007F
-----
```

評価ボードを動かさない
時間があればカウントを始める。
0x7Fでクリップされる。

使用関数

1. void MMA9553_Init(void)
 1. void MMA9553_SetCallBack(void (*pfCallBack)(void))
 2. void (*pfMMA9553CallBack)(void)
 3. void MMA9553_CallBack(void)
 4. uint8_t MMA9553_int(void)
 1. void MMA9553_WakeUp(void)
 2. void MMA9553_Disable(void)
 3. uint8_t MMA9553_Run(void)
 4. void MMA9553_Enable(void)
2. void MMA9553_dump(event_t events)
 1. void MMA9553_Start_Periodical_data(void)
 2. void MMA9553_Periodical_data(uint8_t timerId)
 3. int16_t MMA9553_CatchSensorData(uint8_t number)
 4. uint8_t MMA9553_SetConfig(void)

MMA9553の初期化

```
void MMA9553_Init(void){
```

```
    uint8_t err_cntr;
```

```
    ① MMA9553_SetCallBack(MMA9553_CallBack);  
    err_cntr = 0;
```

```
    ② while( 0x00 != MMA9553_int() ) {  
        err_cntr++;  
    }
```

```
    ③ MMA9553_start_flag = 1;  
    mMMA9553TimerID = TMR_AllocateTimer();  
}
```

定期的にcallする関数を設定するために専用のTimerをallocate。

① call back関数の中でapplication task **AppTask**をcallする設定

```
② uint8_t MMA9553_int(void){  
    uint8_t error = 0;
```

```
    MMA9553_WakeUp();    // Wake up from Deep Sleep  
    MMA9553_Disable();   // Disable the Pedometer application  
    error = MMA9553_Run();
```

```
    if(error==0){  
        MMA9553_Enable();    // Enable the Pedometer application  
    }  
    return error;  
}
```

③ **stateListen** stateになった後, FXAS2100を定期的呼び出すTimer関数をactiveにするflagをセット。

定期的なcallと表示

File: Mapp.cの中の

void AppTask(**event_t** events)

case stateListen:の中で以下の関数が定期的にcallされる。

```
#if gMMA9553_enable
    MMA9553_dump(events);
#endif
```

void FXAS21000_dump(event_t events)

```
void MMA9553_dump(event_t events){
    volatile int16_t tmp;
    uint8_t i;
    // Start to receive periodical data
    if(MMA9553_start_flag){
        MMA9553_start_flag = 0;
        MMA9553_Start_Periodical_data();
    }

    if (events & gAppEvt_FromMMA9553_c)
    {
        UartUtil_Print("\n\r-----\n\r", gAllowToBlock_d);
        for(i=1; i<7; i++){
            switch(i){
                case 1:
                    UartUtil_Print("MMA9553 \n\rX Act_data : 0x", gAllowToBlock_d);
                    break;
                case 2:
                    UartUtil_Print("\n\rX Step_data : 0x", gAllowToBlock_d);
                    break;
                case 3:
                    UartUtil_Print("\n\rX Dist_data : 0x", gAllowToBlock_d);
                    break;
                case 4:
                    UartUtil_Print("\n\rX Speed_data : 0x", gAllowToBlock_d);
                    break;
                case 5:
                    UartUtil_Print("\n\rX Cal_data : 0x", gAllowToBlock_d);
                    break;
                case 6:
                    UartUtil_Print("\n\rX Sleep_cnt_data : 0x", gAllowToBlock_d);
                    break;
                default: break;
            } // switch(i){
            tmp = (uint16_t)MMA9553_CatchSensorData(i);
            UartUtil_PrintHex((uint8_t*)&tmp, 2, 1);
        } // for(i=1; i<7; i++){
        UartUtil_Print("\n\r-----\n\r", gAllowToBlock_d);
    } // if (events & gAppEvt_FromMMA9553_c)
}
```

```
void MMA9553_Start_Periodical_data(void) {
    TMR_StartIntervalTimer( mMMA9553TimerID, mMMA9553Interval_c, MMA9553_Periodical_data );
    =1000ms
}
```

1s定期毎にMMA9553_Periodical_data関数がcallされる。
MMA9553_Periodical_dataは、次頁に記載。

```
int16_t MMA9553_CatchSensorData(uint8_t number){
    volatile int16_t catch_data;
    switch(number){
        case 1:
            catch_data = dataFrom_MMA9553.Act_data;
            break;
        case 2:
            catch_data = dataFrom_MMA9553.Step_data;
            break;
        case 3:
            catch_data = dataFrom_MMA9553.Dist_data;
            break;
        case 4:
            catch_data = dataFrom_MMA9553.Speed_data;
            break;
        case 5:
            catch_data = dataFrom_MMA9553.Cal_data;
            break;
        case 6:
            catch_data = dataFrom_MMA9553.Sleep_cnt_data;
            break;
        default : break;
    }
    return catch_data;
}
```

void MMA9553_Periodical_data(uint8_t timerId) -1

```
void MMA9553_Periodical_data(uint8_t timerId)
{
    volatile int16_t Read_data_16bit = 0;
    volatile uint8_t dumpstatus_data[4]={0x15, 0x30, 0x00, 0x0C};
    volatile uint8_t rxData[20];
    volatile uint8_t check_status = 0;
    uint8_t i;

    (void) timerId; /* prevent compiler warning */
    for(i=0; i<20; i++){ // retry
        IIC_RegWriteN(MMA9553_IIC_ADDRESS,MMA9553_Mailbox,4,(uint8_t *)&dumpstatus_data);
        IIC_RegReadN(MMA9553_IIC_ADDRESS,0x00,16,(uint8_t *)&rxData); // Read data
        check_status = ((rxData[1]&0x80)==0x80) & (rxData[2]==0x0C); // check COCO and num of read bytes
        if(check_status){
            break;    // if no-error
        }
    }
}
```


void MMA9553_Periodical_data(uint8_t timerId) -2

```
while(check_status == 0x00){// if it stays, error occurred..}
```

```
pfMMA9553CallBack(); // Set event in order to notify application in callback function.
```

```
= TS_SendEvent(gAppTaskID_c, gAppEvt_FromMMA9553_c)が実行されAppTaskを call される。
```

```
Read_data_16bit = (int16_t)( rxData[4] & 0x07 );
```

```
dataFrom_MMA9553.Act_data= Read_data_16bit;
```

```
Read_data_16bit = (int16_t)(rxData[6]<<8);
```

```
dataFrom_MMA9553.Step_data = ( Read_data_16bit + (int16_t)rxData[7] );
```

```
Read_data_16bit = (int16_t)(rxData[8]<<8);
```

```
dataFrom_MMA9553.Dist_data = ( Read_data_16bit + (int16_t)rxData[9] );
```

```
Read_data_16bit = (int16_t)(rxData[10]<<8);
```

```
dataFrom_MMA9553.Speed_data = ( Read_data_16bit + (int16_t)rxData[11] );
```

```
Read_data_16bit = (int16_t)(rxData[12]<<8);
```

```
dataFrom_MMA9553.Cal_data = ( Read_data_16bit + (int16_t)rxData[13] );
```

```
Read_data_16bit = (int16_t)(rxData[14]<<8);
```

```
dataFrom_MMA9553.Sleep_cnt_data = ( Read_data_16bit + (int16_t)rxData[15] );
```

```
}
```

void MMA9553_Periodical_data(uint8_t timerId) -3

Buffer	Content
rxData[4]	アクティビティ (4段階)
rxData[6]	歩数[15:8] [steps]
rxData[7]	歩数[7:0] [steps]
rxData[8]	距離[15:8][m]
rxData[9]	距離[7:0][m]
rxData[10]	速度[15:8][m/h]
rxData[11]	速度[7:0][m/h]
rxData[12]	消費カロリー[15:8]
rxData[13]	消費カロリー[7:0]
rxData[14]	Sleepした回数 [15:8][count]
rxData[15]	Sleepした回数 [7:0][count]

パラメータ	Description
dataFrom_MMA9553.Act_data	アクティビティ(4段階)[3bit]
dataFrom_MMA9553.Step_data	歩数[16bit]] [steps]
dataFrom_MMA9553.Dist_data	距離[16bit]] [m]
dataFrom_MMA9553.Speed_data	速度[16bit][m/h]
dataFrom_MMA9553.Cal_data	消費カロリー[16bit]
dataFrom_MMA9553.Sleep_cnt_data	Sleepした回数[16bit][count]

Call back関数の設定

センサconfigurationの中で設定

```
MMA9553_SetCallBack(MMA9553_CallBack)
```

```
void MMA9553_SetCallBack(void (*pfCallBack)(void)) {  
    pfMMA9553CallBack = pfCallBack;  
}
```

```
void MMA9553_CallBack(void){  
    TS_SendEvent(gAppTaskID_c, gAppEvt_FromMMA9553_c);  
    =AppTaskをcallします。 上記Eventをセットします。  
}
```

センサデータを
定期的にread
する関数

```
void MMA9553_Periodical_data(uint8_t timerId ) {  
    pfMMA9553_CallBack(); // Set event in order to notify application in callback function.
```

MMA9553_CallBackを実行

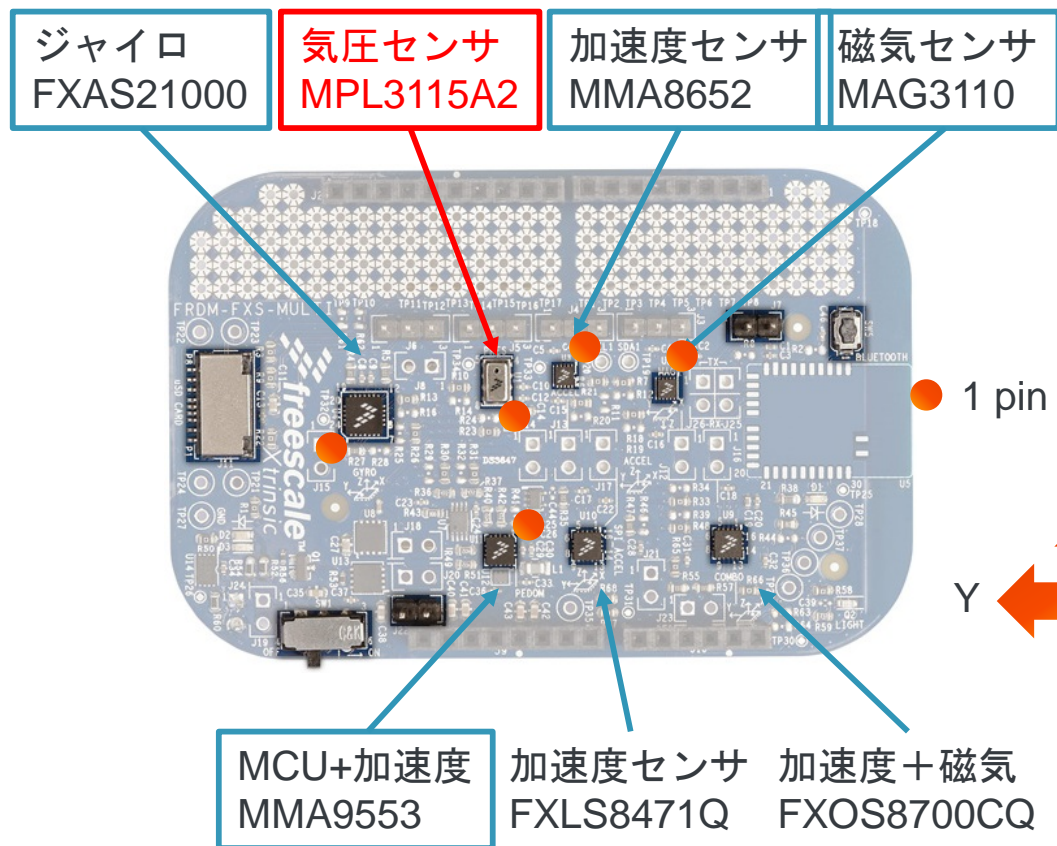
以下の内容を実行

```
void AppTask(event_t events)  
{  
    void MMA9553_dump(event_t events){  
        if (events & gAppEvt_FromMMA9553_c){
```

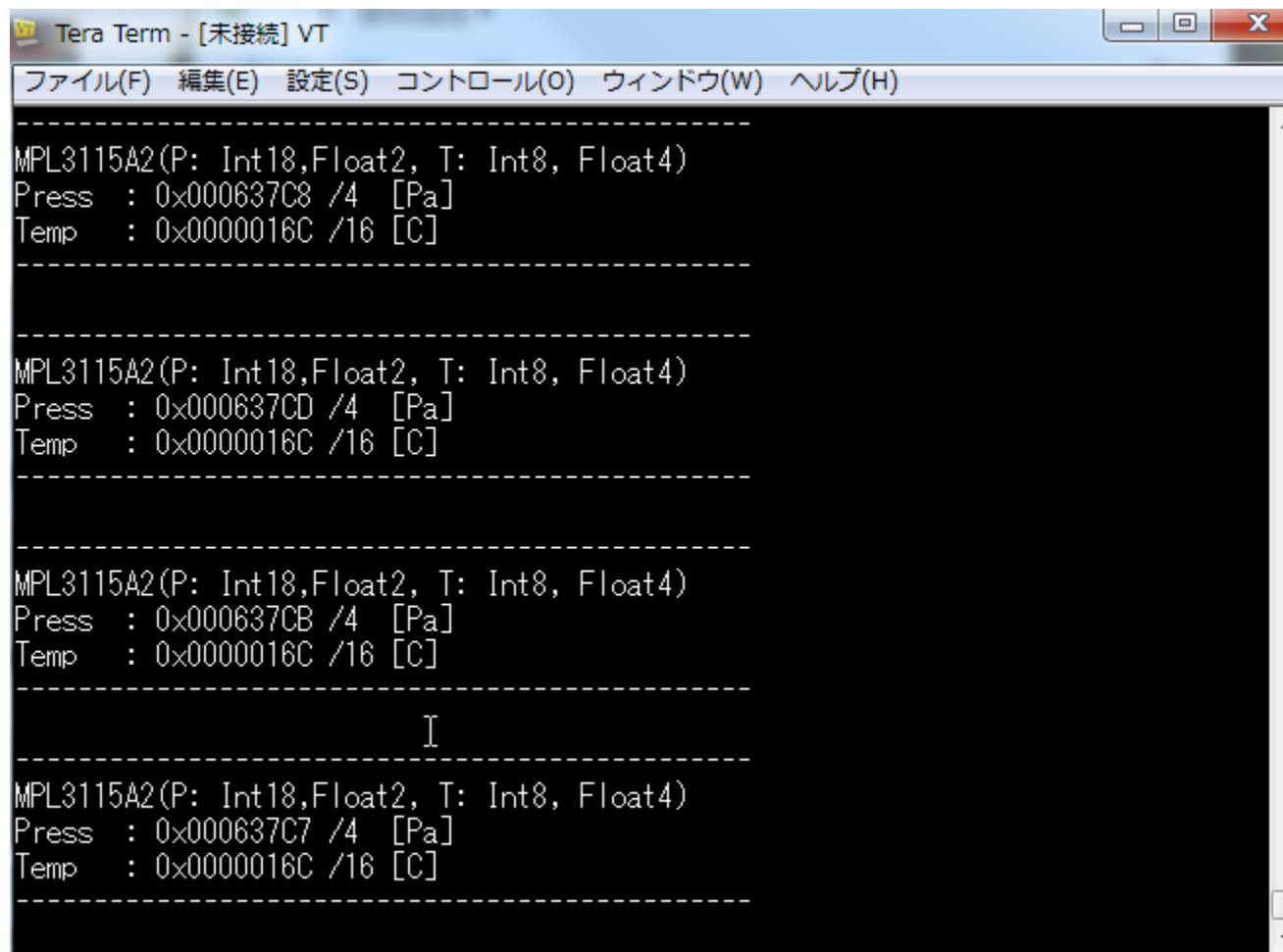


MPL3115A2: 気圧センサ

MPL3115A2: 気圧センサ



MPL3115A2: 気圧センサの表示



The screenshot shows a Tera Term terminal window titled "Tera Term - [未接続] VT". The menu bar includes "ファイル(F)", "編集(E)", "設定(S)", "コントロール(O)", "ウィンドウ(W)", and "ヘルプ(H)". The terminal displays four sets of sensor data, each separated by dashed lines. Each set includes the sensor model "MPL3115A2(P: Int18,Float2, T: Int8, Float4)", the pressure reading "Press : [hex] /4 [Pa]", and the temperature reading "Temp : 0x0000016C /16 [C]". The pressure values are 0x000637C8, 0x000637CD, 0x000637CB, and 0x000637C7. A cursor is visible on the line following the third set of data.

```
Tera Term - [未接続] VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
-----
MPL3115A2(P: Int18,Float2, T: Int8, Float4)
Press : 0x000637C8 /4 [Pa]
Temp  : 0x0000016C /16 [C]
-----
MPL3115A2(P: Int18,Float2, T: Int8, Float4)
Press : 0x000637CD /4 [Pa]
Temp  : 0x0000016C /16 [C]
-----
MPL3115A2(P: Int18,Float2, T: Int8, Float4)
Press : 0x000637CB /4 [Pa]
Temp  : 0x0000016C /16 [C]
-----
                                     I
MPL3115A2(P: Int18,Float2, T: Int8, Float4)
Press : 0x000637C7 /4 [Pa]
Temp  : 0x0000016C /16 [C]
-----
```


使用関

1. **void MPL3115A2_Init(void);**
 1. **void MPL3115A2_Callback(void);**
 2. **void (*pfMPL3115A2Callback)(void);**
 3. **void MPL3115A2_Init_Bar(void);**
2. **void MPL3115A2_dump(event_t events);**
 1. **void MPL3115A2_Active(void);**
 2. **void MPL3115A2_Periodical_data(uint8_t timerId);**
 3. **void MPL3115A2_Start_Periodical_data(void);**
 4. **uint32_t MPL3115A2_CatchSensorData(uint8_t number);**

MPL3115A2の初期化

```
void MPL3115A2_Init(void){
```

```
① MPL3115A2_SetCallBack(MPL3115A2_CallBack);
```

```
② MPL3115A2_Init_Bar();
```

```
③ MPL3115A2_start_flag = 1;
```

```
}
```

- ① call back関数の中でapplication task **AppTask**をcallする設定

- ② `void MPL3115A2_Init_Bar(void) {`

```
IIC_RegWrite(MPL3115A2_SlaveAddressIIC, CTRL_REG1, 0x00); //stand-by mode
```

```
IIC_RegWrite(MPL3115A2_SlaveAddressIIC, PT_DATA_CFG_REG, (DREM_MASK | PDEFE_MASK | TDEFE_MASK));
```

```
IIC_RegWrite(MPL3115A2_SlaveAddressIIC, CTRL_REG1, OSR_128); //Barometer_mode, OSC 512ms
```

```
MPL3115A2_Active();
```

```
mMPL3115A2TimerID = TMR_AllocateTimer();
```

```
}
```

定期的にcallする関数を設定するために専用のTimerをallocate。

- ③ **stateListen** stateになった後, **MPL3115A2**を定期的呼び出すTimer関数をactiveにするflagをセット。

定期的なcallと表示

File: Mapp.cの中の

void AppTask(**event_t** events)

case stateListen:の中で以下の関数が定期的にcallされる。

```
#if gMPL3115A2_enable
    MPL3115A2_dump(events);
#endif
```

void MPL3115A2_dump(event_t events)

```
void MPL3115A2_dump(event_t events){
```

```
    volatile int32_t tmp;  
    volatile uint32_t tmp_unsigned;
```

```
    //-----  
    // Start to receive periodical data  
    if(MPL3115A2_start_flag){  
        MPL3115A2_start_flag = 0;  
        MPL3115A2_Start_Periodical_data();  
    }
```

```
    if (events & gAppEvt_FromMPL3115A2_c)
```

```
    {  
        UartUtil_Print("¥n¥r-----¥n¥r", gAllowToBlock_d);  
        UartUtil_Print("MPL3115A2(P: Int18,Float2, T: Int8, Float4) ¥n¥rPress : 0x", gAllowToBlock_d);  
        tmp = MPL3115A2_CatchSensorData(1);  
  
        tmp_unsigned = (uint32_t)tmp;  
        UartUtil_PrintHex((uint8_t *)&tmp_unsigned, 4, 1);  
  
        UartUtil_Print("/4 [Pa]", gAllowToBlock_d);  
        tmp = MPL3115A2_CatchSensorData(2);  
  
        UartUtil_Print("¥n¥rTemp : 0x", gAllowToBlock_d);  
        tmp_unsigned = (uint32_t)tmp;  
        UartUtil_PrintHex((uint8_t *)&tmp_unsigned, 4, 1);  
        UartUtil_Print("/16 [C]", gAllowToBlock_d);  
        UartUtil_Print("¥n¥r-----¥n¥r", gAllowToBlock_d);  
    }  
}
```

```
void MPL3115A2_Start_Periodical_data(void) {  
    TMR_StartIntervalTimer(mMPL3115A2TimerID, mMPL3115A2Interval_c,MPL3115A2_Periodical_data);  
    =1000ms  
}
```

1s定期毎にMPL3115A2_Periodical_data関数が
callされる。
MPL3115A2_Periodical_dataは、次頁に記載。

```
uint32_t MPL3115A2_CatchSensorData(uint8_t number){  
  
    volatile uint32_t catch_data;  
  
    switch(number){  
        case 1:  
            catch_data = dataFrom_MPL3115A2.Pressure_data;  
            break;  
        case 2:  
            catch_data = dataFrom_MPL3115A2.Temperature_data;  
            break;  
        default :  
            break;  
    }  
  
    return catch_data;  
}
```

void MPL3115A2_Periodical_data(uint8_t timerId)

```
void MPL3115A2_Periodical_data(uint8_t timerId){
```

```
    volatile int32_t Read_data_long = 0;
```

```
    volatile uint8_t rxData[6];
```

```
    int32_t tmp;
```

```
    uint16_t wait_cnt = 0;
```

```
    (void) timerId; /* prevent compiler warning */
```

```
    rxData[0] = IIC_RegRead(MPL3115A2_SlaveAddressIIC, STATUS_00_REG); // checking a STATUS-reg
```

```
    if( rxData[0] & 0x08 ){
```

```
        pfMPL3115A2CallBack(); // Set event in order to notify application in callback function.
```

```
        IIC_RegReadN(MPL3115A2_SlaveAddressIIC, OUT_P_MSB_REG, OUT_T_LSB_REG, (uint8_t *)&rxData[1], 5);
```

```
        if ( rxData[0] & PDR_MASK ){
```

```
            tmp = (int32_t)(rxData[1]);
```

```
            Read_data_long = tmp << 16;
```

```
            Read_data_long = Read_data_long + (int32_t)(rxData[2] << 8);
```

```
            dataFrom_MPL3115A2.Pressure_data = ( Read_data_long + (int32_t)rxData[3] ) >> 4;
```

```
        }
```

```
        if ( rxData[0] & TDR_MASK ){
```

```
            Read_data_long = (int32_t)(rxData[4] << 8);
```

```
            dataFrom_MPL3115A2.Temperature_data = ( Read_data_long + (int32_t)rxData[5] ) >> 4;
```

```
        }
```

```
        IIC_RegWrite(MPL3115A2_SlaveAddressIIC, CTRL_REG1, (OSR_128|OST_MASK));
```

```
    }
```

```
}
```

Buffer	Content
rxData[1]	気圧[19:12][count]/4=[ph]
rxData[2]	気圧[11:4][count]/4=[ph]
rxData[3]	気圧[3:0][count]/4=[ph]
rxData[4]	温度[11:4][count]/16=[°C]
rxData[5]	温度[3:0][count]/16=[°C]

パラメータ	Description
dataFrom_MPL3115A2.Pressure_data	気圧[19:0][count]/4=[ph]
dataFrom_MPL3115A2.Temperature_data	温度[11:0][count]/16=[°C]

Call back関数の設定

センサconfigurationの中で設定

```
MPL3115A2_SetCallBack(MPL3115A2_CallBack);
```

```
void MPL3115A2_SetCallBack(void (*pfCallBack)(void)) {  
    pfMPL3115A2CallBack = pfCallBack;  
}
```

```
void MPL3115A2_CallBack(void){  
    TS_SendEvent(gAppTaskID_c, gAppEvt_FromMPL3115A2_c);  
    =AppTaskをcallします。 上記Eventをセットします。  
}
```

センサデータを
定期的にread
する関数

```
void MPL3115A2_Periodical_data(uint8_t timerId){  
    _pfMPL3115A2CallBack(); // Set event in order to notify application in callback function.
```

MPL3115A2_CallBackを実行

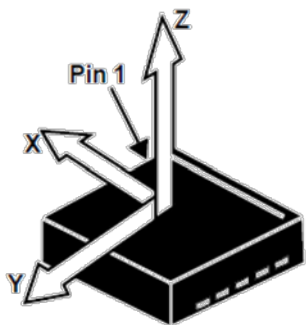
以下の内容を
実行

```
void AppTask(event_t events)  
{  
  
    void MPL3115A2_dump(event_t events){  
        if (events & gAppEvt_FromMPL3115A2_c){
```

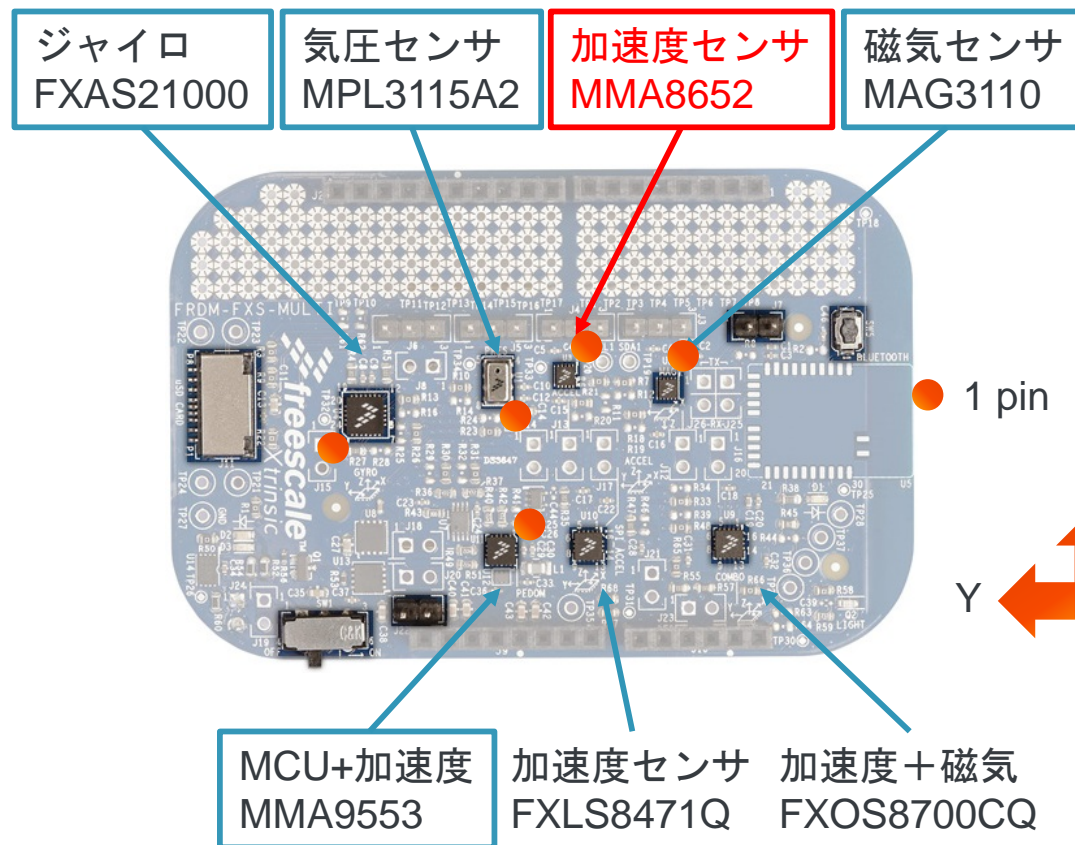



MMA8652: 3軸加速度センサ

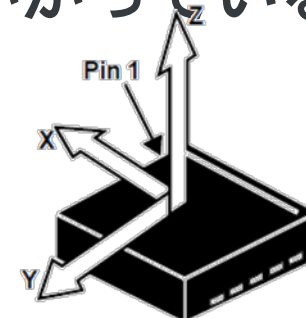
MMA8652: 3軸加速度センサ



MMA8652



MMA8652: 3軸加速度センサの表示 : Z軸にgがかかっている



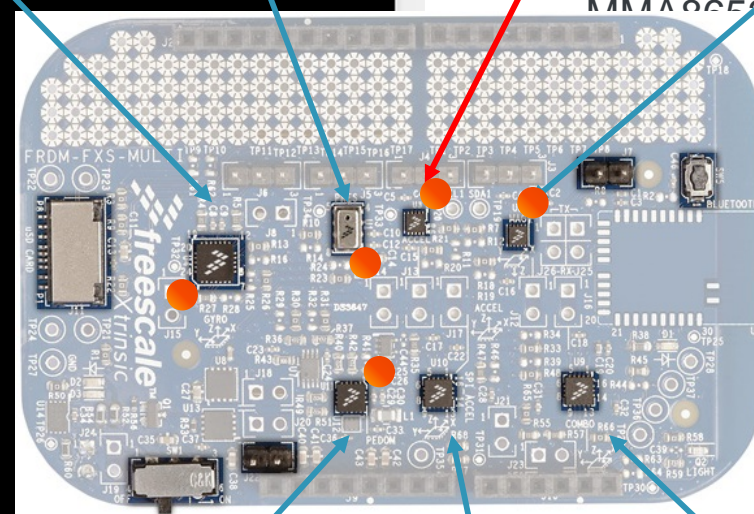
```
Tera Term - [未接続] VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
Z axis : 0x0428 /1024 [g]
-----
MMA8652(2g: Int2,Float10)
X axis : 0x001A /1024 [g]
Y axis : 0xFFFF9 /1024 [g]
Z axis : 0x0425 /1024 [g]
-----
MMA8652(2g: Int2,Float10)
X axis : 0x001B /1024 [g]
Y axis : 0xFFFF8 /1024 [g]
Z axis : 0x0429 /1024 [g]
-----
MMA8652(2g: Int2,Float10)
X axis : 0x001D /1024 [g]
Y axis : 0xFFFF5 /1024 [g]
Z axis : 0x0427 /1024 [g]
-----
```

ジャイロ
FXAS21000

気圧センサ
MPL3115A2

加速度センサ
MMA8652

磁気センサ
MAG3110



MCU+加速度
MMA9553

加速度センサ
FXLS8471Q

加速度+磁気
FXOS8700CQ

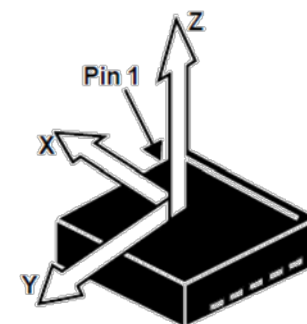
MMA8652: 3軸加速度センサ : Y軸にgがかかっている

```
Tera Term - [未接続] VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

-----
MMA8652(2g: Int2,Float10)
X axis : 0xFFFD /1024 [g]
Y axis : 0x0411 /1024 [g]
Z axis : 0xFF9B /1024 [g]
-----

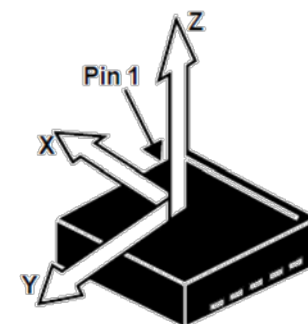
MMA8652(2g: Int2,Float10)
X axis : 0xFF8C /1024 [g]
Y axis : 0x0404 /1024 [g]
Z axis : 0xFF95 /1024 [g]
-----

MMA8652(2g: Int2,Float10)
X axis : 0xFFE9 /1024 [g]
Y axis : 0x0407 /1024 [g]
Z axis : 0xFF60 /1024 [g]
-----
```



MMA8652: 3軸加速度センサ : X軸にgがかかっている

```
Tera Term - [未接続] VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
Z axis : 0x0024 /1024 [g]
-----
MMA8652(2g: Int2,Float10)
X axis : 0x03F2 /1024 [g]
Y axis : 0xFFD5 /1024 [g]
Z axis : 0x0025 /1024 [g]
-----
MMA8652(2g: Int2,Float10)
X axis : 0x03E6 /1024 [g]
Y axis : 0xFFD5 /1024 [g]
Z axis : 0x0024 /1024 [g]
-----
MMA8652(2g: Int2,Float10)
X axis : 0x040A /1024 [g]
Y axis : 0xFFDB /1024 [g]
Z axis : 0x0039 /1024 [g]
-----
```



使用関数

1. void MMA8652_Init(void);
 1. void (*pfMMA8652_Callback)(void);
 2. void MMA8652_SetCallback(void (*pfCallback)(void));
 3. void MMA8652_int(void);
2. void MMA8652_dump(event_t events);
 1. void MMA8652_Start_Periodical_data(void);
 2. void MMA8652_Periodical_data(uint8_t timerId);
 3. int16_t MMA8652_CatchSensorData(uint8_t number);

MMA8652の初期化

```
void MMA8652_Init(void){
```

```
① MMA8652_SetCallBack(MMA8652_CallBack);
```

```
② MMA8652_int();
```

```
③ MMA8652_start_flag = 1;
```

```
}
```

① call back関数の中でapplication task **AppTask**をcallする設定

②

```
void MMA8652_int(void){
```

```
IIC_RegWrite(MMA8652_SlaveAddressIIC,CTRL_REG1,0x00);           // CTRL_REG1: Standby mode
IIC_RegWrite(MMA8652_SlaveAddressIIC,XYZ_DATA_CFG_REG,FULL_SCALE_2G); // XYZ_DATA_CFG:2g mode
IIC_RegWrite(MMA8652_SlaveAddressIIC,CTRL_REG2,MODS1_MASK);      // CTRL_REG2: High Resolution mode
IIC_RegWrite(MMA8652_SlaveAddressIIC,CTRL_REG1,(DR1_MASK|ACTIVE_MASK));
                                                                    // CTRL_REG1: ODR=200Hz(=5ms),Active mode

mMMA8652TimerID = TMR_AllocateTimer();

}
```

定期的にcallする関数を設定するために専用のTimerをallocate。

③ **stateListen** stateになった後, **MMA8652**を定期的呼び出すTimer関数をactiveにするflagをセット。

定期的なcallと表示

File: Mapp.cの中の

void AppTask(**event_t** events)

case stateListen:の中で以下の関数が定期的にcallされる。

```
#if gMMA8652_enable
    MMA8652_dump(events);
#endif
```

void MMA8652_dump(event_t events)

```
void MMA8652_dump(event_t events){
```

```
    volatile int16_t tmp;
```

```
    uint8_t i;
```

```
    // Start to receive periodical data
```

```
    if(MMA8652_start_flag){
```

```
        MMA8652_start_flag = 0;
```

```
        MMA8652_Start_Periodical_data();
```

```
    }
```

```
    if (events & gAppEvt_FromMMA8652_c)
```

```
    {
        UartUtil_Print("\n\r-----\n\r", gAllowToBlock_d);
```

```
        for(i=1; i<4; i++){
```

```
            switch(i){
```

```
                case 1:
```

```
                    UartUtil_Print("MMA8652(2g: Int2,Float10) \n\rX axis : 0x", gAllowToBlock_d);
```

```
                    break;
```

```
                case 2:
```

```
                    UartUtil_Print("/1024 [g]\n\rY axis : 0x", gAllowToBlock_d);
```

```
                    break;
```

```
                case 3:
```

```
                    UartUtil_Print("/1024 [g]\n\rZ axis : 0x", gAllowToBlock_d);
```

```
                    break;
```

```
                default:
```

```
                    UartUtil_Print("????????????", gAllowToBlock_d);
```

```
                    break;
```

```
            }//switch(i){
```

```
            tmp = MMA8652_CatchSensorData(i);
```

```
            UartUtil_PrintHex((uint8_t *)&tmp, 2, 1);
```

```
            if(i==3){
```

```
                UartUtil_Print("/1024 [g]", gAllowToBlock_d);
```

```
            }
```

```
        }//for(i=1; i<4; i++){
```

```
        UartUtil_Print("\n\r-----\n\r", gAllowToBlock_d);
```

```
    }
```

```
}
```

```
void MMA8652_Start_Periodical_data(void){
```

```
    TMR_StartIntervalTimer(mMMA8652TimerID, mMMA8652Interval_c, MMA8652_Periodical_data);
    }
    =700ms
```

1s定期毎にMMA8652_Periodical_data関数がcall
される。
MMA8652_Periodical_dataは、次頁に記載。

```
int16_t MMA8652_CatchSensorData(uint8_t number){
```

```
    int16_t catch_data;
```

```
    switch(number){
```

```
        case 1:
```

```
            catch_data = mDataFrom_MMA8652.xOutReg;
```

```
            break;
```

```
        case 2:
```

```
            catch_data = mDataFrom_MMA8652.yOutReg;
```

```
            break;
```

```
        case 3:
```

```
            catch_data = mDataFrom_MMA8652.zOutReg;
```

```
            break;
```

```
        default :
```

```
            break;
```

```
    }
```

```
    return catch_data;
```

```
}
```

void MMA8652_Periodical_data(timerId)

```
void MMA8652_Periodical_data(uint8_t timerId /* IN: TimerID. */
) {

    uint8_t rxData[7];
    volatile int16_t Read_data_16bit;

    (void) timerId; /* prevent compiler warning */

    rxData[0] = (IIC_RegRead(MMA8652_SlaveAddressIIC,STATUS_00_REG)); // checking a STATUS-reg

    if( rxData[0] & 0x08 ){

        IIC_RegReadN(MMA8652_SlaveAddressIIC,OUT_X_MSB_REG, OUT_Z_LSB_REG, &rxData[1]);
        // Read data from $0x01 to 0x06

        pfMMA8652_CallBack(); // Set event in order to notify application in callback function.

        Read_data_16bit = (int16_t)(rxData[1]<<8);
        mDataFrom_MMA8652.xOutReg = ( Read_data_16bit + (int16_t)rxData[2] ) >>4;

        Read_data_16bit = (int16_t)(rxData[3]<<8);
        mDataFrom_MMA8652.yOutReg = ( Read_data_16bit + (int16_t)rxData[4] ) >>4;

        Read_data_16bit = (int16_t)(rxData[5]<<8);
        mDataFrom_MMA8652.zOutReg = ( Read_data_16bit + (int16_t)rxData[6] ) >>4;
```

Buffer	Content
rxData[1]	X[11:5][count]
rxData[2]	X[4:0][count]
rxData[3]	Y[11:5][count]
rxData[4]	Y[4:0][count]
rxData[5]	Z[11:5][count]
rxData[6]	Z[4:0][count]/

パラメータ	Description
mDataFrom_MMA8652.xOutReg	X[count]/1024=[g]
mDataFrom_MMA8652.yOutReg	Y[count]/1024=[g]
mDataFrom_MMA8652.zOutReg	Z[count]/1024=[g]

Call back関数の設定

センサconfigurationの中で設定

```
MMA8652_SetCallBack(MMA8652_CallBack)
```

```
void MMA8652_SetCallBack(void (*pfCallBack)(void)) {  
    pfMMA8652_CallBack = pfCallBack;  
}
```

```
void MMA8652_CallBack(void){  
    TS_SendEvent(gAppTaskID_c, gAppEvt_FromMMA8652_c);  
    =AppTaskをcallします。 上記Eventをセットします。  
}
```

センサデータを
定期的にread
する関数

```
void MMA8652_Periodical_data(uint8_t timerId ) {  
    pfMMA8652_CallBack(); // Set event in order to notify application in callback function.
```

MMA8652_CallBackを実行

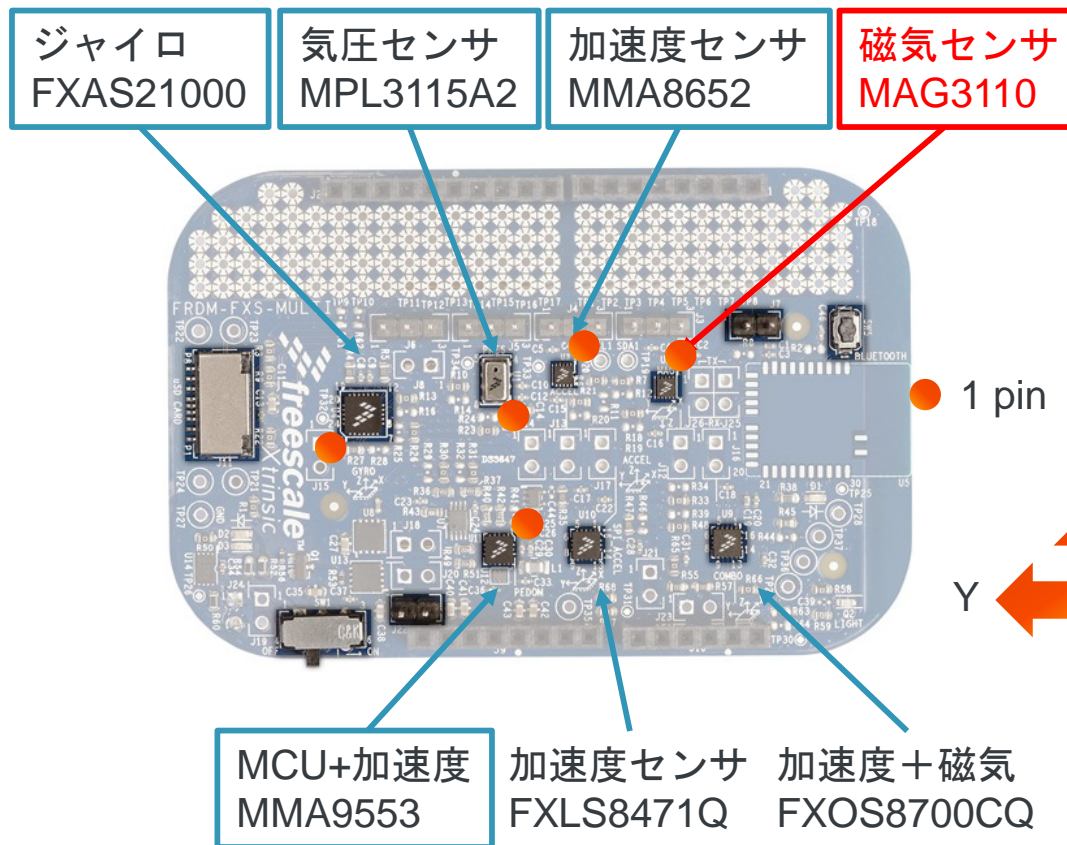
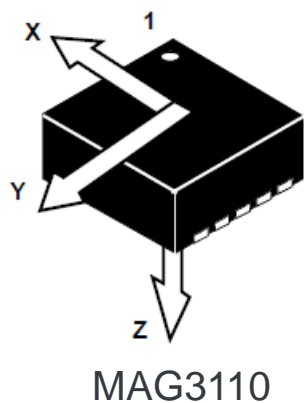
以下の内容を実行

```
void AppTask(event_t events)  
{  
  
    void MMA8652_dump(event_t events){  
  
        if (events & gAppEvt_FromMMA8652_c){
```

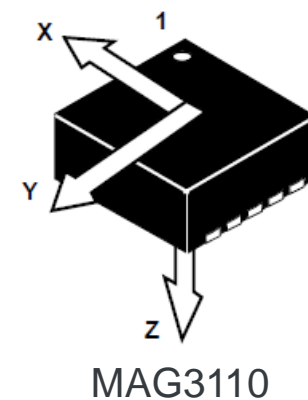
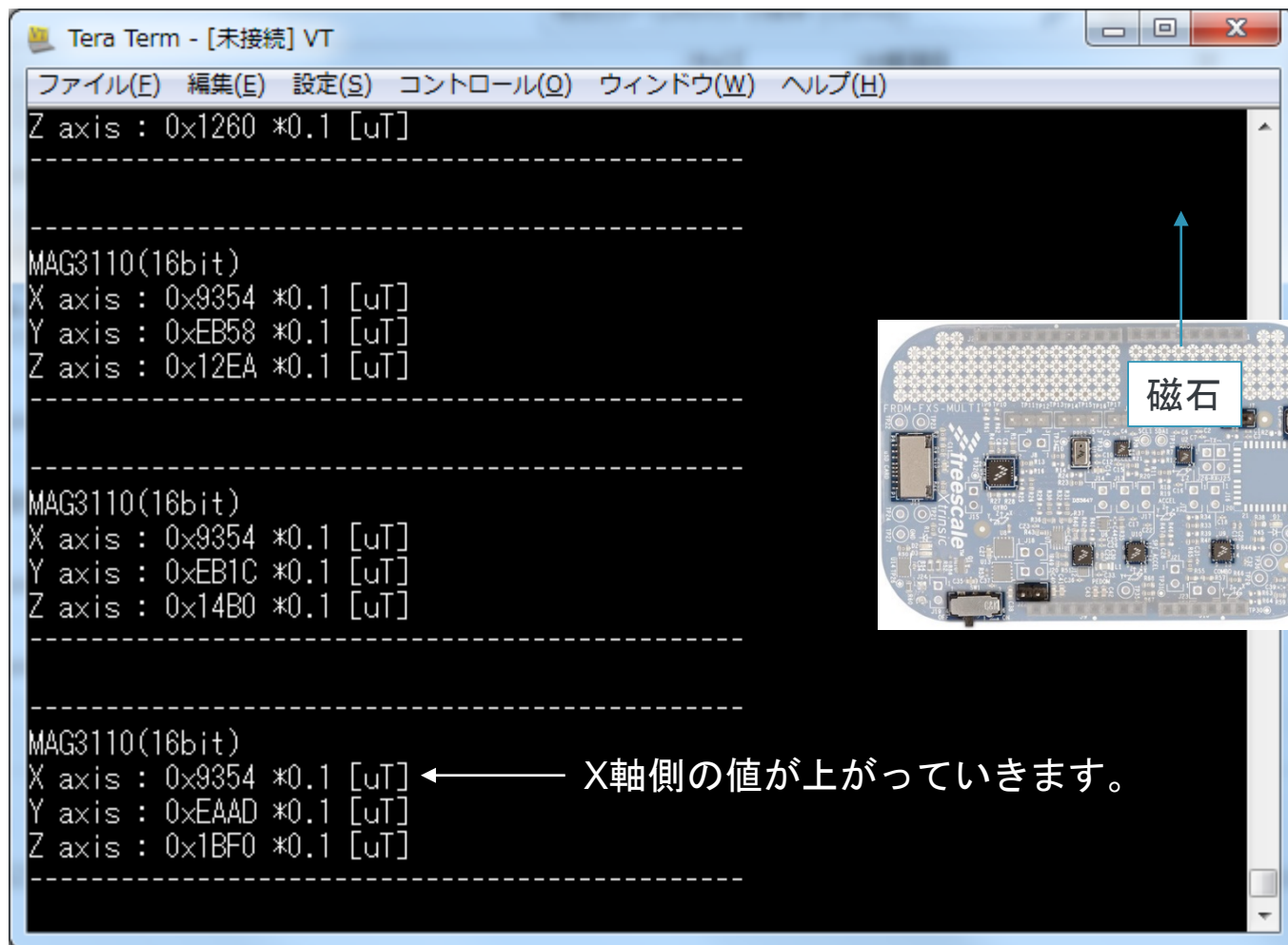


MAG3110: 3軸磁気センサ

MAG3110: 3軸磁気センサ



X軸の値



← X軸側の値が上がっていきます。

Y軸の値

Tera Term - [未接続] VT

ファイル(E) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

Z axis : 0x073B *0.1 [uT]

MAG3110(16bit)

X axis : 0xEE79 *0.1 [uT]

Y axis : 0xDA1C *0.1 [uT]

Z axis : 0x043F *0.1 [uT]

MAG3110(16bit)

X axis : 0xEC63 *0.1 [uT]

Y axis : 0xDD54 *0.1 [uT]

Z axis : 0x011F *0.1 [uT]

MAG3110(16bit)

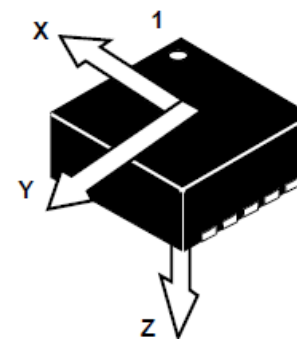
X axis : 0xEC9B *0.1 [uT]

Y axis : 0xDD58 *0.1 [uT]

Z axis : 0x0336 *0.1 [uT]

磁石

Y軸側の値が上がっていきます。



MAG3110

Z軸の値

Tera Term - [未接続] VT

ファイル(F) 編集(E) 設定(S) コントロール(Q) ウィンドウ(W) ヘルプ(H)

Z axis : 0x7FFF *0.1 [uT]

MAG3110(16bit)

X axis : 0x6C1C *0.1 [uT]

Y axis : 0xEECB *0.1 [uT]

Z axis : 0x7FFF *0.1 [uT]

MAG3110(16bit)

X axis : 0x6C16 *0.1 [uT]

Y axis : 0xEE8E *0.1 [uT]

Z axis : 0x7FFF *0.1 [uT]

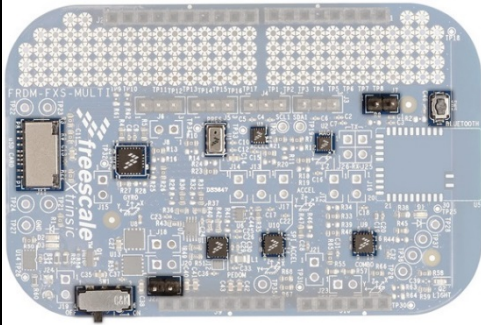
MAG3110(16bit)

X axis : 0x6C1C *0.1 [uT]

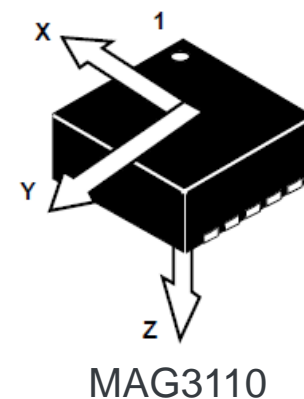
Y axis : 0xEEF1 *0.1 [uT]

Z axis : 0x7FFF *0.1 [uT]

← Z軸側の値が上がっていきます。



磁石
下から



使用関数

```
1. void MAG3110_Init(void)
    1. void (*pfMAG3110_Callback)(void)
    2. void MAG3110_SetCallback(void (*pfCallback)(void))
    3. void MAG3110_Callback(void)
    4. void MAG3110_int(void)

2. void MAG3110_dump(event_t events)
    1. void MAG3110_Start_Periodical_data(void)
    2. void MAG3110_Periodical_data(uint8_t timerId)
    3. int16_t MAG3110_CatchSensorData(uint8_t number)
```

MMA8652の初期化

```
void MAG3110_Init(void){
```

```
① MAG3110_SetCallBack(MAG3110_CallBack);
```

```
② MAG3110_int();
```

```
③ MAG3110_start_flag = 1;
```

```
}
```

- ① call back関数の中でapplication task **AppTask**をcallする設定

② **void MAG3110_int(void){**

```
IIC_RegWrite(MAG3110_SlaveAddressIIC,MAG3110_CTRL_REG1,0x00); // CTRL_REG1: Standby mode
```

```
IIC_RegWrite(MAG3110_SlaveAddressIIC,MAG3110_CTRL_REG2,MAG3110_CTRL_REG2_AUTO_MRST_EN_MASK);
```

```
// CTRL_REG2: RAW=0
```

```
IIC_RegWrite(MAG3110_SlaveAddressIIC,MAG3110_CTRL_REG1,0x19); // CTRL_REG1: Standby mode
```

```
mMAG3110TimerID = TMR_AllocateTimer();
```

```
}
```

定期的にcallする関数を設定するために専用のTimerをallocate。

- ③ **stateListen** stateになった後, **MAG3110**を定期的呼び出すTimer関数をactiveにするflagをセット。

定期的なcallと表示

File: Mapp.cの中の

void AppTask(event_t events)

case *stateListen*:の中で以下の関数が定期的にcallされる。

```
#if gMAG3110_enable
    MAG3110_dump(events);
#endif
```


void MAG3110_dump(event_t events)

```
void MAG3110_dump(event_t events){
```

```
    volatile int16_t tmp;
    uint8_t i;
```

```
    // Start to receive periodical data
    if(MAG3110_start_flag){
        MAG3110_start_flag = 0;
        MAG3110_Start_Periodical_data();
    }
```

```
void MAG3110_Start_Periodical_data(void) {
    TMR_StartIntervalTimer(mMAG3110TimerID, mMAG3110Interval_c, MAG3110_Periodical_data);
    =700ms
}
```

1s定期毎にMAG3110_Periodical_data関数がcallされる。
MAG3110_Periodical_dataは、次頁に記載。

```
    if (events & gAppEvt_FromMAG3110_c){
        UartUtil_Print("¥n¥r-----¥n¥r", gAllowToBlock_d);
        for(i=1; i<4; i++){
            switch(i){
                case 1:
                    UartUtil_Print("MAG3110(16bit) ¥n¥rX axis : 0x", gAllowToBlock_d);
                    break;
                case 2:
                    UartUtil_Print(" *0.1 [uT]¥n¥rY axis : 0x", gAllowToBlock_d);
                    break;
                case 3:
                    UartUtil_Print(" *0.1 [uT]¥n¥rZ axis : 0x", gAllowToBlock_d);
                    break;
                default:
                    UartUtil_Print("?????????????", gAllowToBlock_d);
                    break;
            }
        }
    }
```

```
int16_t MAG3110_CatchSensorData(uint8_t number){
```

```
    int16_t catch_data;
```

```
    switch(number){
        case 1:
            catch_data = mDataFrom_MAG3110.xOutReg;
            break;
        case 2:
            catch_data = mDataFrom_MAG3110.yOutReg;
            break;
        case 3:
            catch_data = mDataFrom_MAG3110.zOutReg;
            break;
        default :
            break;
    }
```

```
    return catch_data;
```

```
}
```

```
    tmp = MAG3110_CatchSensorData(i);
    UartUtil_PrintHex((uint8_t*)&tmp, 2, 1);
```

```
    if(i==3){
        UartUtil_Print(" *0.1 [uT]", gAllowToBlock_d);
    }
    } //for(i=1; i<4; i++){
    UartUtil_Print("¥n¥r-----¥n¥r", gAllowToBlock_d);
}
```

void MAG3110_Periodical_data(timerId)

```
void MAG3110_Periodical_data(uint8_t timerId)
{

uint8_t rxData[7];
volatile int16_t Read_data_16bit;
(void) timerId; /* prevent compiler warning */
rxData[0] = IIC_RegRead(MAG3110_SlaveAddressIIC,MAG3110_DR); // checking a STATUS-reg
if( rxData[0] & 0x08){
    IIC_RegReadN(MAG3110_SlaveAddressIIC,MAG3110_OUT_X_MSB,MAG3110_OUT_Z_LSB,&rxData[1],3);
    pfMAG3110_CallBack(); // Set event in order to notify application in callback function.

    Read_data_16bit = (int16_t)(rxData[1]<<8);
    mDataFrom_MAG3110.xOutReg = ( Read_data_16bit + (int16_t)rxData[2] );

    Read_data_16bit = (int16_t)(rxData[3]<<8);
    mDataFrom_MAG3110.yOutReg = ( Read_data_16bit + (int16_t)rxData[4] );

    Read_data_16bit = (int16_t)(rxData[5]<<8);
    mDataFrom_MAG3110.zOutReg = ( Read_data_16bit + (int16_t)rxData[6] );

}
}
```

Buffer	Content
rxData[1]	X[11:5][count]/1024=[g]
rxData[2]	X[4:0][count]/1024=[g]
rxData[3]	Y[11:5][count]/1024=[g]
rxData[4]	Y[4:0][count]/1024=[g]
rxData[5]	Z[11:5][count]/1024=[g]
rxData[6]	Z[4:0][count]/1024=[g]

パラメータ	Description
mDataFrom_MAG3110.xOutReg	X[count]/1024=[g]
mDataFrom_MAG3110.yOutReg	Y[count]/1024=[g]
mDataFrom_MAG3110.zOutReg	Z[count]/1024=[g]

Call back関数の設定

センサconfigurationの中で設定

```
MAG3110_SetCallBack(MAG3110_CallBack);
```

```
void MAG3110_SetCallBack(void (*pfCallBack)(void)) {  
    pfMAG3110_CallBack = pfCallBack;  
}
```

```
void MAG3110_CallBack(void){  
    TS_SendEvent(gAppTaskID_c, gAppEvt_FromMAG3110_c);  
    =AppTaskをcallします。 上記Eventをセットします。  
}
```

センサデータを
定期的にread
する関数

```
void MAG3110_Periodical_data(uint8_t timerId)  
    pfMAG3110_CallBack(); // Set event in order to notify application in callback function.
```

MAG3110_CallBackを実行

以下の内容を実行

```
void AppTask(event_t events)  
{  
  
    void MAG3110_dump(event_t events){  
        if (events & gAppEvt_FromMAG3110_c){
```



www.Freescale.com