ニューラルソフト有限会社

（公開資料）	技術資料	検認		照査	作成
		―	市来 博記	―	―

表　　題

TDK InvenSense ICM-20948 の使い方

(Raspberry Pi 版)

副　　題

How to use TDK InvenSense ICM-20948 for Rapberry Pi

キ ー ワ ー ド

ICM-20948/SPI/I2C

参照/添付資料

TDK InvenSense HP 内のデータ シートと MPU-9250 から ICM-20948 への移行についてのアプリケーション ノート

スイッチ サイエンス HP 内の Qwiic - ICM-20948 搭載 9DoF IMU のページ

SparkFun HP 内の 9DoF IMU (ICM-20948) Breakout Hookup Guide

GitHub 内の Arduino ライブラリ

Raspberry Pi OS Document

オーダー 番号	オーダー 件名	ソフトウェアー管理番号	改定履歴
----------	----------	----------	~~A B~~ C ~~D E F G H I J K L M N O P Q R S T U V W X Y Z~~

目次

1. はじめに

本書は、TDK InvenSense ICM-20948(モーション トラッキング デバイス)をマイコンに SPI 接続して、加速度、ジャイロ、磁気コンパスの生データとモーション プロセッシングの演算結果を読み取る方法を説明するものです。

2. 用語と略語の定義

本書で使用する用語と略語を表 2-1 に示します。

表 2-1 用語と略語の一覧

用語・略語	説明
SPI	Serial Peripheral Interface モトローラ(現在は NXP セミコンダクターズ)が提唱したシリアル バス規格 比較的低速なデータ転送を行うデバイスとの通信に適している 詳細はウィキペディアを参照のこと
I2C	Inter-Integrated Circuit の略で I-squared-C フィリップス社で開発されたシリアル バス規格 速度よりもシンプル性やコスト面が優先されるデバイスとの通信に適している 詳細はウィキペディアを参照のこと
MEMS	Micro Electro Mechanical Systems 微小な電気機械システム
FSR	フルスケールレンジ
dps	Degree Per Second（度/s）
ホール効果	電流が流れている物体(半導体など)に対して、電流に垂直な磁場をかけると、電流と磁場の両方に垂直な向きに電位差(=起電力)が生じる現象

3. ICM-20948 とは

ICM-20948 は TDK InvenSense 社製の 3 軸 MEMS 加速度センサ、3 軸 MEMS ジャイロセンサ、3 軸シリコン モノリシック ホール効果磁気センサ、デジタル モーション プロセッサ(FPGA)を搭載した超低電力で動作可能な IC です。（この IC にモーション センシングの計算を行わせることが可能です。温度を感知する温度センサも搭載しています。）インターフェースは SPI と I2C をサポートしています。

特長

- 供給電圧：1.71V～3.6V(I/O は 1.8V)
- 3 軸 MEMS 加速度センサ
  FSR　　±2 g、±4 g、±8 g、±16 g
  サンプリング周期、ローパス フィルター等の選択が可能
- 3 軸 MEMS ジャイロセンサ
  FSR　　±250 dps、±500 dps、±1000 dps、±2000 dps
  サンプリング周期、ローパス フィルター等の選択が可能
- 3 軸シリコンモノリシックホール効果磁気センサ(I2C 接続された AK09916 を内蔵)
  FSR　　4900μT（マイクロ テスラ）
  感度　　0.15μT/LSB
- インターフェース
  SPI　　最大 7 MHz
  I2C　　標準モード：最大 100 kHz / 高速モード：最大 400 kHz

4. 前提

本書の記載内容の前提を示します。

- ホスト コントローラは Raspberry Pi 3B を使用します。
- ホスト コントローラの OS は Raspberry Pi OS 32Bit を使用します。
- Raspberry Pi の GPIO はデフォルトのピン割り当てとします。

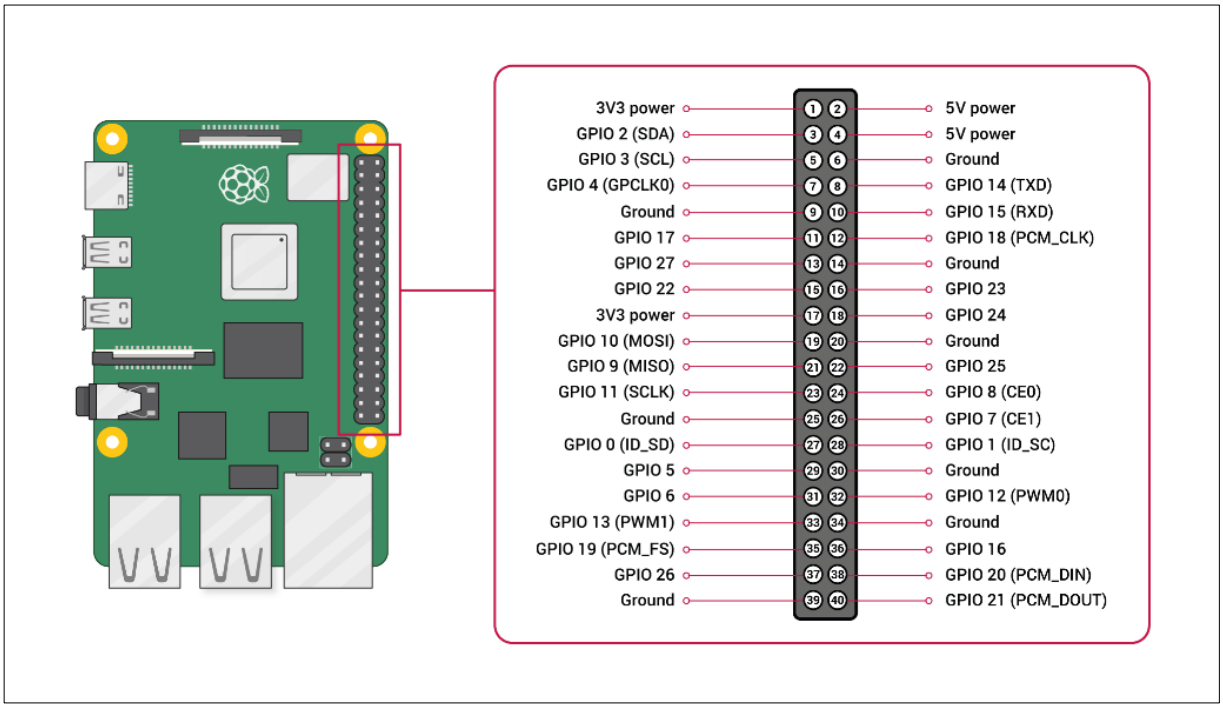


図 4-1 GPIO のピンの割り当て(Raspberry Pi OS Document から引用)

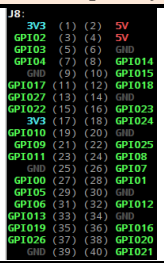
- SparkFun 9DoF IMU Breakout - ICM-20948 ボードを使用します。(電圧変換が不要なので選定しました。)
- ホスト コントローラと SparkFun 9DoF IMU は SPI で接続します。(最初の動作確認のみ I2C)
- ホスト コントローラの I2C デバイス ドライバは linux 標準ドライバ i2c-dev を使用します。
- ホスト コントローラの SPI デバイス ドライバは linux 標準ドライバ spidev を使用します。
- ホスト コントローラで動作させるプログラムの言語は C/C++言語とします。

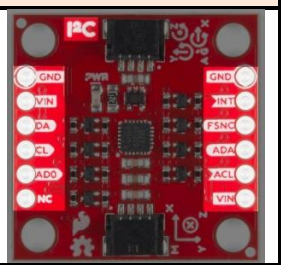
5. 準備

まず最初に I2C で"Who Im I"が取得できるか確認します。(ICM-20948 のレジスタのアドレスとビットの割り付け等の情報はデータ シートに記載されています。)

1. SparkFun 9DoF IMU Breakout - ICM-20948 にピン ヘッダをはんだ付けします。
   (普通の 40 ピンのものをニッパーで切って使用します。)
2. Raspberry Pi の I2C を raspi-config コマンドで有効に設定します。
3. Raspberry Pi の電源を OFF にして、Raspberry Pi と SparkFun 9DoF IMU をジャンパ ワイヤ（メスーメス）で接続します。
   表 5-1 に Raspberry Pi と SparkFun 9DoF IMU の結線を示します。

表 5-1 Raspberry Pi と SparkFun 9DoF IMU の結線(I2C)

Raspberry Pi		結線	SparkFun 9DoF IMU
	1	―	VIN
	9	―	GND
	3	―	DA
	5	―	CL



Raspberry Pi のピン配置図は Raspberry Pi OS Document から引用
SparkFun 9DoF IMU のピン配置図は 9DoF IMU (ICM-20948) Breakout Hookup Guide から引用

4. Raspberry Pi の電源を ON にします。
   SparkFun 9DoF IMU Breakout - ICM-20948 ボード上の電源 LED が点灯します。
5. i2cdetect -l で i2c-1 を確認します。
   以下のように表示されれば OK です。

```
i2c-1   i2c             bcm2835 (i2c@7e804000)                  I2C adapter
```

6. i2cdetect -y 1 で接続状態を確認します。
   以下のように表示されれば OK です。

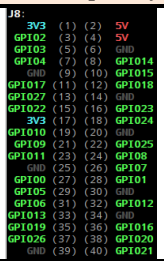
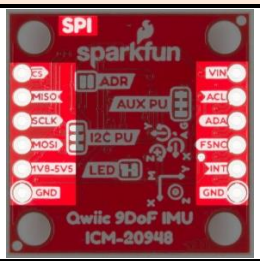
```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 69 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

7. i2cget -y 1 0x69 0x00 で、BANK0 の”Who Im I”を取得します。
   取得値が 0xEA となれば OK です。
8. i2cset -y 1 0x69 0x7f 0x30 b で BANK を 3 に切り替えます。
9. i2cget -y 1 0x69 0x7f で、0x30 が取得値となれば OK です。
10. i2cset -y 1 0x69 0x7f 0x00 b で BANK を 0 に切り替えて、7 を実行します。

SPI 通信の準備に移ります。

1. Raspberry Pi の SPI を raspi-config コマンドで有効に設定します。
   （I2C を使わない場合は、無効に設定します。）
2. Raspberry Pi の電源を OFF にして、I2C 用に接続したジャンパ ワイヤを SPI 接続に変更します。
   表 5-2 に Raspberry Pi と SparkFun 9DoF IMU の結線を示します。

表 5-2 Raspberry Pi と SparkFun 9DoF IMU の結線(SPI)

Raspberry Pi		結線	SparkFun 9DoF IMU	
	1	―	VIN	
	9	―	GND	
	24	―	$\overline{CS}$	
	21	―	MISO	
	23	―	SCLK	
	19	―	MOSI	

Raspberry Pi のピン配置図は Raspberry Pi OS Document から引用
SparkFun 9DoF IMU のピン配置図は 9DoF IMU (ICM-20948) Breakout Hookup Guide から引用

3. Raspberry Pi の電源を ON にします。
   SparkFun 9DoF IMU Breakout - ICM-20948 ボード上の電源 LED が点灯します。
4. lsmod | grep spi でモジュールがロードされていることを確認します。
   以下のように表示されれば OK です。

```
spidev                  20480  0
spi_bcm2835             20480  0
```

5. ls -l /dev | grep spi で SPI0 のデバイスファイルが作成されていることを確認します。

```
crw-rw----  1 root spi     153,   0 xx 月  xx xx:xx spidev0.0
crw-rw----  1 root spi     153,   1 xx 月  xx xx:xx spidev0.1
```

6. これで、open 関数と ioctl 関数を使用してデバイスと通信することが出来ます。
   以下に open 関数と ioctl 関数の使用例を示します。

デバイスをオープン

```
int fd = open("/dev/spidev0.0", O_RDWR | O_SYNC);          // open の戻り値が負の場合はエラー
```

クロックの極性と位相の設定

```
constexpr UInt8 mode = SPI_MODE_3;                         // モード番号についてはウィキペディアを参照
UInt8 wMode = mode;
UInt8 rMode = 0;
int sts = ioctl(fd, SPI_IOC_WR_MODE, &wMode);              // ioctl の戻り値が負の場合はエラー
    sts = ioctl(fd, SPI_IOC_RD_MODE, &rMode);              // ioctl の戻り値が負の場合はエラー
                                                           //   rMode = wMode | 0x04 となる
```

ワード長を設定

```
constexpr UInt8 bits = 8;
UInt8 wBits = bits;
UInt8 rBits = 0;
int sts = ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &wBits);     // ioctl の戻り値が負の場合はエラー
    sts = ioctl(fd, SPI_IOC_RD_BITS_PER_WORD, &rBits);     // ioctl の戻り値が負の場合はエラー
```

ビット順を設定

```
constexpr UInt8 first = 0;                                 // LSB-first の場合は SPI_LSB_FIRST
UInt8 wFirst = first;
UInt8 rFirst = 0;
int sts = ioctl(fd, SPI_IOC_WR_LSB_FIRST, &wFirst);        // ioctl の戻り値が負の場合はエラー
    sts = ioctl(fd, SPI_IOC_RD_LSB_FIRST, &rFirst);        // ioctl の戻り値が負の場合はエラー
```

最高クロック周波数を設定

```
constexpr UInt32 speed = 4*1000*1000;                      // 4MHz(ICM-20948 の最高周波数は 7MHz)
UInt32 wMaxSpeed = speed;
UInt32 rMaxSpeed = 0;
int sts = ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &wMaxSpeed);  // ioctl の戻り値が負の場合はエラー
    sts = ioctl(fd, SPI_IOC_RD_MAX_SPEED_HZ, &rMaxSpeed);  // ioctl の戻り値が負の場合はエラー
```

レジスタ リード

```
constexpr UInt32 transferSize = 2;              // サイズを大きくすれば、その分バースト転送できる
UInt8 buff[transferSize] = { 0x00 | 0x80 };
struct spi_ioc_transfer tr = {
       .tx_buf = UInt64(buff),
       .rx_buf = UInt64(buff),                  // 受信バッファに送信バッファを指定できる(上書きされる)
       .len = transferSize,
       .speed_hz = speed,                       // 今回の転送に適用するクロック周波数
       .delay_usecs = 0,                        // 転送終了から CS を NACT にするまでのディレイ時間
       .bits_per_word = bits,                   // 今回の転送に適用するワード長
       .cs_change = 0                           // true の場合、MESSAGE 毎に CS を NACT/ACT する
                                                // (最後は ACT のままになる)
};
int sts = ioctl(fd, SPI_IOC_MESSAGE(1), &tr);   // sts は転送サイズ(負の場合はエラー)
                                                // レジスタ値は buff[1]に格納される
```

レジスタ ライト

```
constexpr UInt32 transferSize = 2;              // サイズを大きくすれば、その分バースト転送できる
UInt8 buff[transferSize] = { 0x7f, 0x30 };
struct spi_ioc_transfer tr = {
       .tx_buf = UInt64(buff),
       .rx_buf = 0,
       .len = transferSize,
       .speed_hz = speed,                       // 今回の転送に適用するクロック周波数
       .delay_usecs = 0,                        // 転送終了から CS を NACT にするまでのディレイ時間
       .bits_per_word = bits,                   // 今回の転送に適用するワード長
       .cs_change = 0                           // true の場合、MESSAGE 毎に CS を NACT/ACT する
                                                // (最後は ACT のままになる)
};
int sts = ioctl(fd, SPI_IOC_MESSAGE(1), &tr);   // sts は転送サイズ(負の場合はエラー)
                                                // レジスタ値は buff[1]に格納される
```

7. 慣例や好みに合わせて SPI 通信用ラップ クラスを作成します。

この資料を書く発端となったプロジェクトで調査用に作成したラップ クラスのヘッダと、ラップクラスを使用して”Who Im I”の受信と BANK 3 への切り替えを確認するテスト コードを示します。

[SPI 通信用ラップ クラス ヘッダ]

```
#include <linux/spi/spidev.h>

namespace rbxSys
{
    class SPI
    {
        FD mDev;    //  SPI デバイスのファイル ディスクリプタ
        UInt8 mBitsPerWord;
        UInt32 mMaxFreqHz;

    public:
        //  クロック モード
        static constexpr UInt8 CLK_MODE0 = SPI_MODE_0;
        static constexpr UInt8 CLK_MODE1 = SPI_MODE_1;
        static constexpr UInt8 CLK_MODE2 = SPI_MODE_2;
        static constexpr UInt8 CLK_MODE3 = SPI_MODE_3;

        //  構築
        SPI(UInt8 port, UInt8 chip);
        //  破棄
        virtual ~SPI();

        //  利用可能確認
        Bool IsAvalableSPI(void) const {
            return mDev >= 0;
        }
        //  セット アップ
        Bool SetupSPI(
            UInt8 clockMode, UInt8 bitsPerWord,
            Bool bitOrderLSB = false, UInt32 maxFreqHz = 1 * 1000 * 1000);

        //  レジスタ リード
        UInt8 ReadReg(UInt8 addr) const;
        //  レジスタ ライト
        void WriteReg(UInt8 addr, UInt8 v) const;

        Int32 WriteReadBlocking(const UInt8* wBuff, UInt8* rBufft, UInt32 len) const;
        Int32 WriteBlocking(const UInt8* buff, UInt32 len) const;
        Int32 ReadBlocking(UInt8* buff, UInt32 len) const;
    };
}
```

["Who Im I"の受信と BANK 3 への切り替えを確認するテスト コード]

```
          :
          :
rbxSys::SPI spi(0, 0);
Bool sts = spi.SetupSPI(rbxSys::SPI::CLK_MODE3, 8);
if (sts == true)
{
    //  バンク 0 に切替
    spi.WriteReg(0x7f, 0x00);
    //  ID リード
    UInt8 d = spi.ReadReg(0x00);          // d が 0xEA なら OK
    //  バンク 3 に切替
    spi.WriteReg(0x7f, 0x30);
    //  アドレス 0x7f をリード
    d = spi.ReadReg(0x7f);                // d が 0x30 なら OK
    //  バンク 0 に切替
    spi.WriteReg(0x7f, 0x00);
    //  アドレス 0x00 をリード
    d = spi.ReadReg(0x00);                // d が 0xEA なら OK
          :
          :
}
          :
          :
```

準備はここまでです。後は ICM-20948 の初期化処理、起動処理、データ取得処理の作成なのですが、ネット上に、処理の手順に関する資料を見つけることが出来ませんでした。仕方なく SparkFun 9DoF IMU の Arduino ライブラリ[1]を解析しましたので、概要を次章に記載します。

[1] SparkFun 9DoF IMU 用の ArduinoLibrary は、SparkFun 9DoF IMU (ICM-20948) Breakout Hookup Guide のページ、又は GitHub からダウンロードできます。Arduino 以外のプラットフォーム用の ICM-20948 のサンプル コードは、TDK InvenSense の Software Download のページからダウンロードできます。

6. 処理手順

ICM-20948 を利用するソフトウェアーの処理の手順を示します。FSR、サンプリング、ローパス フィルタの設定値は Arduino ライブラリのコードの内容ですので、利用目的に合わせて変更する必要があります。(SPI の通信の処理については前章の 6 項を参照して下さい。)

(1) ICM-20948 の生データのみを取得して、MPU で補正演算を行う場合の初期化処理

(各処理が正常に完了しない場合は、エラー ログを出力して処理を中止するようになっています。)

1 ICM-20948 のチップ ID 確認
- 1.1 BANK0 を選択(BANK の選択方法は A の処理を参照)
- 1.2 WHO_AM_I (0x00)レジスタをリード
- 1.3 WHO_AM_I が 0xEA か確認

2 ソフトウェアー リセットを実行して、ICM-20948 を初期状態に移行
- 2.1 BANK0 を選択
- 2.2 PWR_MGMT_1 (0x06)レジスタの DEVICE_RESET を 1 に設定

3 50 ミリ秒ウェイト

4 スリープ解除
- 4.1 BANK0 を選択
- 4.2 AGB0_REG_PWR_MGMT_1(0x06)レジスタの SLEEP を 0 に設定

5 省電力モード解除
- 5.1 BANK0 を選択
- 5.2 AGB0_REG_PWR_MGMT_1(0x06)レジスタの LP_EN を 0 に設定

6 磁気コンパス(AK09916)起動
- 6.1 I2C マスタ有効化
  - 6.1.1 I2C マスタ パス スルーを無効化
    - 6.1.1.1 BANK0 を選択
    - 6.1.1.2 AGB0_REG_INT_PIN_CONFIG(0x0F)レジスタの BYPASS_EN を 0 に設定
  - 6.1.2 BANK3 を選択
  - 6.1.3 AGB3_REG_I2C_MST_CTRL(0x01)レジスタの I2C_MST_CLK を 0x07、I2C_MST_P_NSR を 1 に設定
  - 6.1.4 BANK0 を選択
  - 6.1.5 AGB0_REG_USER_CTRL(0x03)レジスタの I2C_MST_EN を 1 に設定
- 6.2 磁気コンパス リセット
  - 6.2.1 AK09916_REG_CNTL3(0x32)レジスタの SRST を 1 に設定(B の処理を参照)
- 6.3 磁気コンパスが応答するまで磁気コンパスをリセット(最大 10 回トライ)
  - 6.3.1 磁気コンパスの ID を確認
    - 6.3.1.1 AK09916_REG_WIA1 レジスタをリード(B の処理を参照)
    - 6.3.1.2 AK09916_REG_WIA2 レジスタをリード(B の処理を参照)
    - 6.3.1.3 AK09916_REG_WIA1 レジスタ値 << 8 | AK09916_REG_WIA2 レジスタ値が 0x4809 か確認
  - 6.3.2 磁気コンパスの ID を確認完了の場合は、6.4 に進む
  - 6.3.3 I2C マスタ リセット
    - 6.3.3.1 BANK0 を選択
    - 6.3.3.2 AGB0_REG_USER_CTRL(0x03)レジスタの I2C_MST_RST を 1 に設定
  - 6.3.4 10 ミリ秒後に 6.3.1 に戻る
- 6.4 磁気コンパス設定
  - 6.4.1 AK09916_REG_CNTL2 レジスタ(0x31)の MODE に mode_cont_100hz(0x04 << 1)を設定(B の処理を参照)
  - 6.4.2 磁気コンパス レジスタ転送設定(C の処理を参照)

    ペリフェラル番号=0,I2C アドレス=0x0C,レジスタ アドレス=AK09916_REG_ST1(0x10),転送サイズ=9

    リード/ライト指定=リード,有効指定=有効,データのみ指定=false,ワードのグループ指定=false,バイト スワップ指定=false

7 サンプリング モード設定
- 7.1 BANK0 を選択
- 7.2 AGB0_REG_LP_CONFIG(0x05)レジスタに次の値を設定

  .ACCEL_CYCLE=SMPL_MODE_CONTINUOUS(0)

  .GYRO_CYCLE=SMPL_MODE_CONTINUOUS(0)

8 フル スケール レンジ設定
- 8.1 BANK2 を選択
- 8.2 AGB2_REG_ACCEL_CONFIG(0x14)レジスタの ACCEL_FS_SEL に ACCEL_FSS_2G(0)を設定
- 8.3 AGB2_REG_GYRO_CONFIG_1(0x01)レジスタの GYRO_FS_SEL に GYRO_FSS_250DPS(0)を設定

9 デジタル ローパス フィルター設定
- 9.1 BANK2 を選択
- 9.2 AGB2_REG_ACCEL_CONFIG(0x14)レジスタの ACCEL_DLPFCFG に ACCEL_d473bw_n499bw(0x07)を設定

  (dAbwB_nXbwZ : A ⇒ 3db 帯域の整数部, B ⇒ 3db 帯域の小数部

  X ⇒ ナイキスト帯域幅の整数部, Y ⇒ ナイキスト帯域幅の小数部)
- 9.3 AGB2_REG_GYRO_CONFIG_1(0x01)レジスタの GYRO_DLPFCFG に GYRO_d361bw4_n376bw5(0x07)を設定

  (dAbwB_nXbwZ : A ⇒ 3db 帯域の整数部, B ⇒ 3db 帯域の小数部

  X ⇒ ナイキスト帯域幅の整数部, Y ⇒ ナイキスト帯域幅の小数部)

10　デジタル ローパス フィルター有効化

10.1　BANK2 を選択
10.2　AGB2_REG_ACCEL_CONFIG(0x14)レジスタの ACCEL_FCHOICE に ACCEL_DLPF_BYPASS(0x00)を設定
10.3　AGB2_REG_GYRO_CONFIG_1(0x01)レジスタの GYRO_FCHOICE に GYRO_DLPF_DLPF_BYPASS(0x00)を設定

A.　＜BANK 選択(バンク)＞

A.1　バンクが既に選択済みの場合は処理を完了
A.2　REG_BANK_SEL(0x7F)レジスタの USER_BANK に指定のバンク番号を設定

B.　＜磁気コンパス レジスタ ライト(AK09916 のレジスタ アドレス, ライト値)＞

B.1　BANK3 を選択
B.2　AGB3_REG_I2C_PERIPH4_ADDR(0x13)レジスタに MAG_AK09916_I2C_ADDR(0x0C)を設定
B.3　AGB3_REG_I2C_PERIPH4_REG(0x14)レジスタに AK09916 のレジスタ アドレスを設定
B.4　AGB3_REG_I2C_PERIPH4_DO(0x16)レジスタにライト値を設定
B.5　AGB3_REG_I2C_PERIPH4_CTRL(0x15)レジスタに次の値をライトして、AK09916 への転送を起動
　　.EN=1、.INT_EN=0、.DLY = 0、.REG_DIS = 0
B.6　BANK0 を選択
B.7　AGB0_REG_I2C_MST_STATUS(0x17)レジスタをリード(最大 1000 回トライ)して、
　　I2C_PERIPH4_DONE が 1、I2C_PERIPH4_NACK が 0 となれば正常終了、
　　I2C_PERIPH4_DONE が 1、I2C_PERIPH4_NACK が 1 となれば異常終了、
　　I2C_PERIPH4_DONE が 1 にならなければ、異常終了

B.　＜磁気コンパス レジスタ リード(AK09916 のレジスタ アドレス)＞

B.1　BANK3 を選択
B.2　AGB3_REG_I2C_PERIPH4_ADDR(0x13)レジスタに MAG_AK09916_I2C_ADDR(0x0C)を設定
B.3　AGB3_REG_I2C_PERIPH4_REG(0x14)レジスタに AK09916 のレジスタ アドレス | 0x80 を設定
B.4　AGB3_REG_I2C_PERIPH4_CTRL(0x15)レジスタに次の値をライトして、AK09916 からの転送を起動
　　.EN=1、.INT_EN=0、.DLY = 0、.REG_DIS = 0
B.5　BANK0 を選択
B.6　AGB0_REG_I2C_MST_STATUS(0x17)レジスタをリード(最大 1000 回トライ)して、
　　I2C_PERIPH4_DONE が 1、I2C_PERIPH4_NACK が 0 となれば次の処理に遷移
　　I2C_PERIPH4_DONE が 1、I2C_PERIPH4_NACK が 1 となれば異常終了、
　　I2C_PERIPH4_DONE が 1 にならなければ、異常終了
B.7　BANK3 を選択
B.8　AGB3_REG_I2C_PERIPH4_DI(0x17)レジスタを戻り値とする

C.　＜I2C ペリフェラル データ転送設定(
　　ペリフェラル番号, I2C アドレス, レジスタ アドレス, 転送サイズ,
　　リード/ライト指定, 有効指定, データのみ指定, ワードのグループ指定,バイト スワップ指定,
　　ライト値)＞

C.1　ペリフェラル番号からレジスタのアドレスを求める
　　I2C_PERIPH_ADDR = I2C_SLV(x)_ADDR(0x03/0x07/0x0B/0x0F)
　　I2C_PERIPH_REG = I2C_SLV(x)_REG (0x04/0x08/0x0C/0x10)
　　I2C_PERIPH_CTRL = I2C_SLV(x)_CTRL(0x05/0x09/0x0D/0x11)
　　I2C_PERIPH_DO   = I2C_SLV(x)_DO (0x06/0x0A/0x0E/0x12)
C.2　BANK3 を選択
C.3　I2C_PERIPH_ADDR レジスタに I2C アドレスを設定
C.4　リード/ライト指定がライトの場合は I2C_PERIPH_DO レジスタにライト値を設定
C.5　I2C_PERIPH_REG レジスタにレジスタ アドレスを設定
C.6　I2C_PERIPH_CTRL レジスタに次の値を設定
　　.LENG=転送サイズ、ctrl.EN=有効指定、REG_DIS=データのみ指定、
　　.GRP=ワードのグループ指定、.BYTE_SW=バイト スワップ指定

(2) ICM-20948 の生データのみを取得して、MPU で補正演算を行う場合の生データ読み出し処理

（各処理が正常に完了しない場合は、エラー ログを出力して処理を中止するようになっています。）

1　データ レディ待ち

1.1　BANK0 を選択

1.2　AGB0_REG_INT_STATUS_1(0x1A)レジスタの RAW_DATA_0_RDY_INT が 1 の場合は 2 に進む

1.3　500 ミリ秒ウェイトした後、1.1 に戻る

2　生データ取得

2.1　BANK0 を選択

2.2　AGB0_REG_ACCEL_XOUT_H から EXT_SLV_SENS_DATA_08 までバースト リード(14+9 バイト)

磁気コンパス(AK09916)のデータは以下の通り

EXT_SLV_SENS_DATA_00：ST1

EXT_SLV_SENS_DATA_01：HXL

EXT_SLV_SENS_DATA_02：HXH

EXT_SLV_SENS_DATA_03：HYL

EXT_SLV_SENS_DATA_04：HYH

EXT_SLV_SENS_DATA_05：HZL

EXT_SLV_SENS_DATA_06：HZH

EXT_SLV_SENS_DATA_07：TMPS(Dummy)

EXT_SLV_SENS_DATA_08：ST2

2.3　加速度データ(X/Y/Z)、ジャイロ データ(X/Y/Z)、温度データを Int16 型の変数に格納(Lo/Hi スワップ)

2.4　EXT_SLV_SENS_DATA_00：ST1 を UInt8 型の変数に格納

2.5　磁気コンパス(X/Y/Z)データを Int16 型の変数に格納(Lo/Hi スワップなし)

2.6　EXT_SLV_SENS_DATA_08：ST2 を UInt8 型の変数に格納

2.7　BANK2 を選択

2.8　AGB2_REG_ACCEL_CONFIG(0x14)の ACCEL_FS_SEL を UInt8 型のビット フィールドに格納

2.9　AGB2_REG_GYRO_CONFIG_1(0x01)の GYRO_FS_SEL を UInt8 型のビット フィールドに格納

2.10　Int16 型の加速度データ(X/Y/Z)を FLoat32 型(mG)に変換

ACCEL_FS_SEL	mG
0(±2g)	Float32(Int16 値)×Inv(16.384)
1(±4g)	Float32(Int16 値)×Inv(8.192)
2(±8g)	Float32(Int16 値)×Inv(4.096)
3(±16g)	Float32(Int16 値)×Inv(2.048)

2.11　Int16 型のジャイロ データ(X/Y/Z)を FLoat32 型(DPS)に変換

GYRO_FS_SEL	DPS
0(±250dps)	Float32(Int16 値)×Inv(131.0)
1(±500dps)	Float32(Int16 値)×Inv(65.5)
2(±1000dps)	Float32(Int16 値)×Inv(32.8)
3(±2000dps)	Float32(Int16 値)×Inv(16.4)

2.12　Int16 型の温度データを Float32 型(℃)に変換

温度(℃) = Float32(Int16 値 - 21)×Inv(333.87F) + 21.0F

2.13　Int16 型の磁気コンパス データ(X/Y/Z)を FLoat32 型(μT)に変換

磁束密度(μT) = Float32(Int16 値)×0.15

3　生データを表示

4　30 ミリ秒ウェイト後、1 に戻る

---

(3) ICM-20948 の DMP の演算結果を利用する場合の初期化処理

（各処理が正常に完了しない場合は、エラー ログを出力して処理を中止するようになっています。）

1　(1)項の 1 から 6.3 まで同じ

2　デジタル モーション プロセッサの初期化

2.1　磁気コンパス レジスタ転送(ペリフェラル 0)設定((1)項の C の処理を参照)
ペリフェラル番号=0, I2C アドレス=0x0C, レジスタ アドレス=AK09916_REG_RSV2(0x03), 転送サイズ=10
リード/ライト指定=リード, 有効指定=有効, データのみ指定=false, ワードのグループ指定=true, バイト スワップ指定=true

2.2　磁気コンパス レジスタ転送(ペリフェラル 1)設定((1)項の C の処理を参照)
ペリフェラル番号=0, I2C アドレス=0x0C, レジスタ アドレス=AK09916_REG_CNTL2(0x31), 転送サイズ=1
リード/ライト指定=ライト, 有効指定=有効, データのみ指定=false, ワードのグループ指定=true, バイト スワップ指定=true
ライト値=Single measurement mode(0x01)

2.3　I2C マスターODR(Output Data Rate)設定

2.3.1　BANK3 を選択

2.3.2　AGB3_REG_I2C_MST_ODR_CONFIG(0x00) レジスタの I2C_MST_ODR_CONFIG に 0x04 を設定
ODR は 1.1kHz÷$2^{I2C\_MST\_ODR\_CONFIG}$=68.75Hz となる

2.4　クロック ソース設定

2.4.1　BANK0 を選択

2.4.2　AGB0_REG_PWR_MGMT_1(0x06) レジスタの CLKSEL に Clock_Auto(0x01)を設定

2.5　加速度・ジャイロ センサ有効化

2.5.1　BANK0 を選択

2.5.2　AGB0_REG_PWR_MGMT_2(0x07)に 0x40 を設定
加速度とジャイロ センサの全軸をオン
（データ シート上はリザーブ ビットを 1 にしている。）

2.6　磁気コンパスのサンプル モード設定

2.6.1　BANK0 を選択

2.6.2　AGB0_REG_LP_CONFIG(0x05) レジスタの I2C_MST_CYCLE に SMPL_MODE_CYCLED(0x01)を設定

2.7　FIFO 無効化(A の処理を参照)

2.8　DMP 無効化(D の処理を参照)

2.9　フル スケール レンジ設定

2.9.1　BANK2 を選択

2.9.2　AGB2_REG_ACCEL_CONFIG(0x14) レジスタの ACCEL_FS_SEL に ACCEL_FSS_4G(0x01)を設定

2.9.3　AGB2_REG_GYRO_CONFIG_1(0x01) レジスタの GYRO_FS_SEL に GYRO_FSS_2000DPS(0x03)を設定

2.10　デジタル ローパス フィルター設定

2.10.1 BANK2 を選択

2.10.2 AGB2_REG_ACCEL_CONFIG(0x14) レジスタの ACCEL_DLPFCFG に ACCEL_d246bw_n265bw(0x00)を設定

2.10.3 AGB2_REG_GYRO_CONFIG_1(0x01) レジスタの GYRO_DLPFCFG に GYRO_d196bw6_n229bw8(0x00)を設定

2.11　デジタル ローパス フィルター有効化

2.11.1 BANK2 を選択

2.11.2 AGB2_REG_ACCEL_CONFIG(0x14) レジスタの ACCEL_FCHOICE に ACCEL_DLPF_ENABLE(0x01)を設定

2.11.3 AGB2_REG_GYRO_CONFIG_1(0x01) レジスタの GYRO_FCHOICE に GYRO_DLPF_DLPF_ENABLE(0x01)を設定

2.12　//FIFO オーバー フロー割り込みを有効化(コメント アウト)

2.12.1 //BANK0 を選択

2.12.2 //AGB0_REG_INT_ENABLE_2(0x12) レジスタの FIFO_OVERFLOW_EN_0～4 に 1 を設定

2.13　ペリフェラルの FIFO を無効化

2.13.1 BANK0 を選択

2.13.2 AGB0_REG_FIFO_EN_1(0x66) レジスタの PERIPH_0_FIFO_EN～PERIPH_3_FIFO_EN に 0 を設定

2.14　センサ データの FIFO を無効化

2.14.1 BANK0 を選択

2.14.2 AGB0_REG_FIFO_EN_2(0x67) レジスタに次の値を設定
. ACCEL_FIFO_EN=0, . GYRO_X_FIFO_EN=0, . GYRO_Y_FIFO_EN=0, . GYRO_Z_FIFO_EN=0, . TEMP_FIFO_EN=0

2.15　データ レディ割り込み無効化

2.15.1 BANK0 を選択

2.15.2 AGB0_REG_INT_ENABLE_1(0x11) レジスタの RAW_DATA_0_RDY_EN に 0 を設定

2.15.3 AGB0_REG_INT_ENABLE_1(0x11) レジスタの RAW_DATA_0_RDY_EN が 0 でない場合はエラーとする

2.16　FIFO リセット(B の処理を参照)

2.17　サンプル レート設定(加速度：56.25Hz / ジャイロ：55Hz)
// 1.125 kHz/(1+ACCEL_SMPLRT_DIV[11:0])
// 1.1　kHz/(1+GYRO_SMPLRT_DIV [7:0])
// ACCEL_SMPLRT_DIV[11:0] == 19 -> 56.25Hz
// GYRO_SMPLRT_DIV [7:0] == 19 -> 55Hz
// ACCEL_SMPLRT_DIV[11:0] == 4 -> 225Hz
// GYRO_SMPLRT_DIV [7:0] == 4 -> 225Hz
// ACCEL_SMPLRT_DIV[11:0] == 5 -> 112Hz
// GYRO_SMPLRT_DIV [7:0] == 5 -> 112Hz

2.17.1 BANK2 を選択

2.17.2 AGB2_REG_ACCEL_SMPLRT_DIV_1(0x10) レジスタの ACCEL_SMPLRT_DIV[11:8]に UInt16(19)の上位 4 ビットを設定

2.17.3 AGB2_REG_ACCEL_SMPLRT_DIV_2(0x11) レジスタに ACCEL_SMPLRT_DIV[7:0]に UInt16(19)の下位 8 ビットを設定

2.17.4 AGB2_REG_GYRO_SMPLRT_DIV(0x00) レジスタの GYRO_SMPLRT_DIV に 0x19 を設定

2.18　DMP スタート アドレス(0x1000)設定(C の処理を参照)
2.19　ファームウェア ロード(D の処理を参照)
2.20　DMP スタート アドレス(0x1000)設定(C の処理を参照)
2.21　ハードウェア フィックス無効レジスタ設定
  2.21.1 BANK0 を選択
  2.21.2 AGB0_REG_HW_FIX_DISABLE(0x75)レジスタに 0x48 を設定
2.22　シングル FIFO プライオリティ設定
  2.22.1 BANK0 を選択
  2.22.2 AGB0_REG_SINGLE_FIFO_PRIORITY_SEL(0x26)レジスタに 0xE4 を設定
2.23　DMP 加速度データのスケーリングを構成
```
// FSR 1g -> 2^25
// FSR 4g -> 0x04000000
ACC_SCALE : 30 * 16 + 0
accScale = BYTE{ 0x04, 0x00, 0x00, 0x00 }
E の処理(ACC_SCALE, sizeof(accScale), accScale)
```
2.24　DMP 加速度データのスケーリング 2 を構成
```
// FSR 1g -> 2^17
// FSR 4g -> 0x00040000
ACC_SCALE2 : 79 * 16 + 4
accScale2 = BYTE{ 0x00, 0x04, 0x00, 0x00 }
E の処理(ACC_SCALE2, sizeof(accScale2), accScale2)
```
2.25　コンパス マウント マトリクス設定
```
// コンパス マウント マトリクス設定
// コンパス マウント マトリクスはコンパスの軸を加速/ジャイロ センサに合わせるために使用される
// このメカニズムはハードウェア ユニットを uT(マイクロ テスラ)に変換するためにも使用される(1uT = 2^30)
// 各コンパス軸は次のように変換される
//   X = raw_x * CPASS_MTX_00 + raw_y * CPASS_MTX_01 + raw_z * CPASS_MTX_02
//   Y = raw_x * CPASS_MTX_10 + raw_y * CPASS_MTX_11 + raw_z * CPASS_MTX_12
//   Z = raw_x * CPASS_MTX_20 + raw_y * CPASS_MTX_21 + raw_z * CPASS_MTX_22
// AK09916 は ±4912uT に対応する±32752 の範囲で 16 ビットの符号付きデータを出力する(1uT = 6.66ADU)
// 2^30 / 6.66666 = 161061273 = 0x9999999
CPASS_MTX_00:23 * 16          CPASS_MTX_01:23 * 16 + 4      CPASS_MTX_02: 23 * 16 + 8
CPASS_MTX_10:23 * 16 + 12     CPASS_MTX_11:24 * 16          CPASS_MTX_12: 24 * 16 + 4
CPASS_MTX_20:24 * 16 + 8      CPASS_MTX_21:24 * 16 + 12     CPASS_MTX_22: 25 * 16
mountMulZero  = BYTE{ 0x00, 0x00, 0x00, 0x00 }
mountMulPlus  = BYTE{ 0x09, 0x99, 0x99, 0x99 }
mountMulMinus = BYTE{ 0xF6, 0x66, 0x66, 0x67 }
E の処理(CPASS_MTX_00, sizeof(mountMulPlus), mountMulPlus)
E の処理(CPASS_MTX_01, sizeof(mountMulZero), mountMulZero)
E の処理(CPASS_MTX_02, sizeof(mountMulZero), mountMulZero)
E の処理(CPASS_MTX_10, sizeof(mountMulZero), mountMulZero)
E の処理(CPASS_MTX_11, sizeof(mountMulMinus), mountMulMinus)
E の処理(CPASS_MTX_12, sizeof(mountMulZero), mountMulZero)
E の処理(CPASS_MTX_20, sizeof(mountMulZero), mountMulZero)
E の処理(CPASS_MTX_21, sizeof(mountMulZero), mountMulZero)
E の処理(CPASS_MTX_22, sizeof(mountMulMinus), mountMulMinus)
```
2.26　B2S マウント マトリクス設定
```
B2S_MTX_00 : 208 * 16          B2S_MTX_01 : 208 * 16 + 4     B2S_MTX_02 : 208 * 16 + 8
B2S_MTX_10 : 208 * 16 + 12     B2S_MTX_11 : 209 * 16         B2S_MTX_12 : 209 * 16 + 4
B2S_MTX_20 : 209 * 16 + 8      B2S_MTX_21 : 209 * 16 + 12    B2S_MTX_22 : 210 * 16
b2sMountMulZero  = BYTE{ 0x00, 0x00, 0x00, 0x00 }
b2sMountMulPlus  = BYTE{ 0x40, 0x00, 0x00, 0x00 }
E の処理(B2S_MTX_00, sizeof(b2sMountMulPlus), b2sMountMulPlus)
E の処理(B2S_MTX_01, sizeof(b2sMountMulZero), b2sMountMulZero)
E の処理(B2S_MTX_02, sizeof(b2sMountMulZero), b2sMountMulZero)
E の処理(B2S_MTX_10, sizeof(b2sMountMulZero), b2sMountMulZero)
E の処理(B2S_MTX_11, sizeof(b2sMountMulPlus), b2sMountMulPlus)
E の処理(B2S_MTX_12, sizeof(b2sMountMulZero), b2sMountMulZero)
E の処理(B2S_MTX_20, sizeof(b2sMountMulZero), b2sMountMulZero)
E の処理(B2S_MTX_21, sizeof(b2sMountMulZero), b2sMountMulZero)
E の処理(B2S_MTX_22, sizeof(b2sMountMulPlus), b2sMountMulPlus)
```
2.27　ジャイロ スケーリング ファクター設定(G の処理(19 = 55Hz, 3 = 2000dps)を参照)
2.28　ジャイロ フル スケール設定
```
// 2000dps : 2^28
// 1000dps : 2^27
//  500dps : 2^26
//  250dps : 2^25
GYRO_FULLSCALE : 72 * 16 + 12
gyroFullScale = BYTE{ 0x10, 0x00, 0x00, 0x00 } // 2000dps : 2^28
E の処理(GYRO_FULLSCALE, sizeof(gyroFullScale), gyroFullScale)
```

2.29　加速度ゲイン設定
// 225Hz : 15252014
// 112Hz : 30504029
// 56Hz : 61117001
ACCEL_ONLY_GAIN : 16 * 16 + 12
accelOnlyGain = BYTE{ 0x03, 0xA4, 0x92, 0x49 }
Eの処理(ACCEL_ONLY_GAIN, sizeof(accelOnlyGain), accelOnlyGain)
2.30　加速度α設定
// 225Hz : 1026019965
// 112Hz : 977872018
// 56Hz : 882002213
ACCEL_ALPHA_VAR : 91 * 16
accelAlphaVar = BYTE{ 0x34, 0x92, 0x49, 0x25 }
Eの処理(ACCEL_ALPHA_VAR, sizeof(accelAlphaVar), accelAlphaVar)
2.31　加速度A設定
// 225Hz : 47721859
// 112Hz : 95869806
// 56Hz : 191739611
ACCEL_A_VAR : 92 * 16
accelAVar = BYTE{ 0x0B, 0x6D, 0xB6, 0xDB }
Eの処理(ACCEL_A_VAR, sizeof(accelAVar), accelAVar)
2.32　加速度補正レート設定
// 0 (225Hz, 112Hz, 56Hz)
ACCEL_CAL_RATE : 94 * 16 + 4
accelCalRate = BYTE{ 0x00, 0x00 } // Value taken from InvenSense Nucleo example
Eの処理(ACCEL_CAL_RATE, sizeof(accelCalRate), accelCalRate)
2.33　磁気コンパス タイム バッファ設定
// I2Cマスター ODR(Output Data Rate)を 68.75Hz に設定したので、Compass Time Buffer を 69Hz に設定
CPASS_TIME_BUFFER : 112 * 16 + 14
compassRate = BYTE{ 0x00, 0x45 } // 69Hz
Eの処理(CPASS_TIME_BUFFER, sizeof(compassRate), compassRate)
2.34　// DMP割り込み有効化(コメント アウト)
2.34.1 //BANK0を選択
2.34.2 // AGB0_REG_INT_ENABLE(0x10)レジスタの DMP_INT1_EN に1を設定
2.34.3 // AGB0_REG_INT_ENABLE(0x10)レジスタの DMP_INT1_EN が1でない場合、異常終了

A.　<FIFO有効化(有効指定[有効=0x01/無効=0x00])>
A.1　BANK0を選択
A.2　AGB0_REG_USER_CTRL(0x03)レジスタの FIFO_EN に有効指定を設定

B.　<FIFOリセット>
B.1　BANK0を選択
B.2　AGB0_REG_FIFO_RST(0x68)レジスタの FIFO_RESET に 0x1F を設定
B.3　AGB0_REG_FIFO_RST(0x68)レジスタの FIFO_RESET に 0x1E を設定

C.　<DMP スタート アドレス設定(アドレス[デフォルト=0x1000])>
C.1　BANK2を選択
C.2　AGB2_REG_PRGM_START_ADDRH(0x50-0x51)レジスタにアドレスを設定(0x50=上位 / 0x51=下位)

D.　<DMP ファーム ウェア ロード>
D.1　スリープ解除
D.1.1　BANK0を選択
D.1.2　AGB0_REG_PWR_MGMT_1(0x06)レジスタの SLEEP を0に設定
D.2　省電力モード解除
D.2.1　BANK0を選択
D.2.2　AGB0_REG_PWR_MGMT_1(0x06)レジスタの LP_EN を0に設定
D.3　アドレス0x90から始まる領域へ icm20948_img.dmp3a.h のテーブルを16バイトずつバースト ライト
(16バイトのバースト ライトは Eの処理を参照)
D.4　アドレス0x90から始まる領域から16バイトずつバースト リードして、icm20948_img.dmp3a.h のテーブルと比較
(一致しない場合は異常終了)(16バイトのバースト リードは Fの処理を参照)

E. <DMP Mems ライト(ライト アドレス(転送先),ライト サイズ,転送元アドレス)>
E.1 BANK0 を選択
E.2 AGB0_REG_MEM_BANK_SEL(0x7E)レジスタにライト アドレスの上位バイトを設定
E.3 ライト サイズ分繰り返し
E.3.1 AGB0_REG_MEM_START_ADDR(0x7C)レジスタにライト アドレスの下位バイト(バンク内オフセット)を設定
E.3.2 AGB0_REG_MEM_R_W アドレスに最大 16 バイト バースト ライト
E.3.3 ライト サイズとライト アドレスを更新

F. <DMP Mems リード(リード アドレス(転送元), リード サイズ,転送先アドレス)>
F.1 BANK0 を選択
F.2 AGB0_REG_MEM_BANK_SEL(0x7E)レジスタにリード アドレスの上位バイトを設定
F.3 リード サイズ分繰り返し
F.3.1 AGB0_REG_MEM_START_ADDR(0x7C)レジスタにリード アドレスの下位バイト(バンク内オフセット)を設定
F.3.2 (AGB0_REG_MEM_R_W | 0x80)アドレスから最大 16 バイト バースト リード
F.3.3 リード サイズとリード アドレスを更新

G. <ジャイロ スケーリング ファクター設定(サンプリング分周値,レベル)>
// サンプリング分周値(GYRO_SMPLRT_DIV レジスタへの設定値)
// 0=1125Hz サンプル レート, 1=562.5Hz, ... 4=225Hz, ...
// 10=102.2727Hz, ...
// レベル 0=250dps, 1=500dps, 2=1000dps, 3=2000dps
G.1 レベルを 4 に変更(API dmp_icm20648_set_gyro_fsr()が追加された為)
G.2 BANK1 を選択
G.3 変数 pll ← AGB1_REG_TIMEBASE_CORRECTION_PLL(0x28)レジスタをリード(符号付き(通常の値は 0x18))
G.4 変数 sf ← ジャイロ スケーリング ファクター値を算出(Int64 で計算して、0x7FFFFFFF でトリムして Int32 にキャスト)
pll の最上位ビットが 1 の場合の計算式()
(MagicConstant * Int64(1ULL << レベル) * (1 + div) / (1270 - (pll & 0x7F)) / MagicConstantScale)
pll の最上位ビットが 0 の場合の計算式(Int64 で計算)
(MagicConstant * Int64(1ULL << レベル) * (1 + div) / (1270 + pll) / MagicConstantScale)
尚、MagicConstant と MagicConstantScale の値は以下とする
MagicConstant = 264446880937391LL
MagicConstantScale = 100000LL
“参考 InvenSense example inv_icm20948_set_gyro_sf()”のコメントあり
G.5 DMPGYRO_SF メモリ書き込み
GYRO_SF : 19 * 16
dmpGyroSF = BYTE{ sf >> 24, sf >> 16, sf >> 8, sf & 0xff }
E の処理(GYRO_SF, sizeof(dmpGyroSF), dmpGyroSF)

H. <DMP 有効化(有効指定[有効=0x01/無効=0x00])>
H.1 BANK0 を選択
H.2 AGB0_REG_USER_CTRL(0x03)レジスタの DMP_EN に有効指定を設定

I. <DMP リセット>
I.1 BANK0 を選択
I.2 AGB0_REG_USER_CTRL(0x03)レジスタの DMP_RST に 1 を設定

(4) ICM-20948 の DMP の演算結果を利用する場合の起動処理

（各処理が正常に完了しない場合は、エラー ログを出力して処理を中止するようになっています。）

1 センサ有効化(A の処理を参照)
2 センサ データ出力レート設定(B の処理を参照)
3 FIFO 有効化((3)項の A の処理を参照)
4 DMP 有効化((3)項の H の処理を参照)
5 DMP リセット((3)項の I の処理を参照)
6 FIFO リセット((3)項の B の処理を参照)

A. ＜センサ有効化(センサ番号, 有効/無効指定[1:有効/0:無効])＞

```
// センサ番号(括弧内の番号はアンドロイド センサ番号)
//   0 : ACCELEROMETER(1)                  10 : ROTATION_VECTOR(11)                  20 : RAW_MAGNETOMETER(44)
//   1 : GYROSCOPE(4)                      11 : GEOMAGNETIC_ROTATION_VECTOR(20)
//   2 : RAW_ACCELEROMETER(42)             12 : GEOMAGNETIC_FIELD(2)
//   3 : RAW_GYROSCOPE(43)                 13 : WAKEUP_SIGNIFICANT_MOTION(17)
//   4 : MAGNETIC_FIELD_UNCALIBRATED(14)   14 : FLIP_PICKUP(46)
//   5 : GYROSCOPE_UNCALIBRATED(16)        15 : WAKEUP_TILT_DETECTOR(41)
//   6 : ACTIVITY_CLASSIFICATON(47)        16 : GRAVITY(9)
//   7 : STEP_DETECTOR(18)                 17 : LINEAR_ACCELERATION(10)
//   8 : STEP_COUNTER(19)                  18 : ORIENTATION(3)
//   9 : GAME_ROTATION_VECTOR(15)          19 : B2S(45)
```

A.1 センサ番号をアンドロイド センサ番号に変換(上記のセンサ番号の説明を参照)

A.2 アンドロイド センサ番号が 44 以上の場合は異常終了

A.3 delta(変更する出力データのビット パターン) ← アンドロイド センサ番号を出力データ ビット パターンに変換

```
//出力データ ビット パターンの定義 (0xFFFF は非サポートを表す)
// 0x8000 : 16-bit accel                                0x0080 : 16-bit Pressure
// 0x4000 : 16-bit gyro                                 0x0040 : 32-bit calibrated gyro
// 0x2000 : 16-bit compass                              0x0020 : 32-bit calibrated compass
// 0x1000 : 16-bit ALS                                  0x0010 : Pedometer Step Detector
// 0x0800 : 32-bit 6-axis quaternion                    0x0008 : Header 2
// 0x0400 : 32-bit 9-axis quaternion + heading accuracy 0x0004 : Pedometer Step Indicator Bit 2
// 0x0200 : 16-bit pedometer quaternion                 0x0002 : Pedometer Step Indicator Bit 1
// 0x0100 : 32-bit Geomag rv + heading accuracy         0x0001 : Pedometer Step Indicator Bit 0

// アンドロイド センサ番号と出力データ ビット パターンの対応(アンダーバー付きのアンドロイド センサ番号名は先頭に"WAKEUP_"が付く)
// アンドロイド センサ番号          ビット パターン  アンドロイド センサ番号          ビット パターン  アンドロイド センサ番号      ビット パターン
//  0:META_DATA                   0xFFFF
//  1:ACCELEROMETER               0x8008  23:ACCELEROMETER              0x8008  41:TILT_DETECTOR      0x0000
//  2:MAGNETIC_FIELD              0x0028  24:MAGNETIC_FIELD             0x0028  42:Raw Acc            0x8008
//  3:ORIENTATION                 0x0408  25:ORIENTATION                0x0408  43:Raw Gyr            0x4048
//  4:GYROSCOPE                   0x4048  26:GYROSCOPE                  0x4048
//  5:LIGHT                       0x1008  27:LIGHT                      0x1008
//  6:PRESSURE                    0x0088  28:PRESSURE                   0x0088
//  7:TEMPERATURE                 0xFFFF
//  8:PROXIMITY                   0xFFFF
//  9:GRAVITY                     0x0808  29:GRAVITY                    0x0808
// 10:LINEAR_ACCELERATION         0x8808  30:LINEAR_ACCELERATION        0x8808
// 11:ROTATION_VECTOR             0x0408  31:ROTATION_VECTOR            0x0408
// 12:HUMIDITY                    0xFFFF  32:RELATIVE_HUMIDITY          0xFFFF
// 13:AMBIENT_TEMPERATURE         0xFFFF  33:AMBIENT_TEMPERATURE        0xFFFF
// 14:MAGNETIC_FIELD_UNCAL        0x2008  34:MAGNETIC_FIELD_UNCAL       0x2008
// 15:GAME_ROTATION_VECTOR        0x0808  35:GAME_ROTATION_VECTOR       0x0808
// 16:GYROSCOPE_UNCAL             0x4008  36:GYROSCOPE_UNCAL            0x4008
// 17:SIGNIFICANT_MOTION          0x0000
// 18:STEP_DETECTOR               0x0018  37:STEP_DETECTOR              0x0018
// 19:STEP_COUNTER                0x0010  38:STEP_COUNTER               0x0010
// 20:GEO_ROTATION_VECTOR         0x0108  39:GEO_ROTATION_VECTOR        0x0108
// 21:HEART_RATE                  0xFFFF  40:HEART_RATE                 0xFFFF
// 22:PROXIMITY                   0xFFFF
```

A.4 delta が 0xFFFF の場合は異常終了

A.5 アンドロイド センサ番号が 32 より小さい場合、有効/無効指定に応じて android0 のアンドロイド センサ番号をビット位置とするビットを更新
(動作中は android0 は保持されるので、有効なセンサ(アンドロイド センサ番号<32)に対応するビットが全て 1 になる)

A.6 アンドロイド センサ番号が 32 以上場合、有効/無効指定に応じて android1 のアンドロイド センサ番号-32 をビット位置とするビットを更新
(動作中は android1 は保持されているので、有効なセンサ(アンドロイド センサ番号≧32)に対応するビットが全て 1 になる)

A.7　出力データ ビット パターン、データ レディ ステータス、イベント制御を確定
A.7.1　delta(出力データ ビット パターン),data_rdy_status(データ レディ ステータス),inv_event_control(イベント制御)をクリア
A.7.2　delta ← android0 と android1 で有効になっている全センサのデータ出力ビット パターンを求める
A.7.3　android0 又は android1 で有効になっている全センサの内、加速度データを出力するセンサがある場合、
　　data_rdy_status　|= DMP_Data_ready_Accel(0x0002)
　　inv_event_control |= DMP_Motion_Event_Control_Accel_Calibr(0x0200)
A.7.4　android0 又は android1 で有効になっている全センサの内、ジャイロ データを出力するセンサがある場合、
　　data_rdy_status　|= DMP_Data_ready_Gyro(0x0001)
　　inv_event_control |= DMP_Motion_Event_Control_Gyro_Calibr(0x0100)
A.7.5　android0 又は android1 で有効になっている全センサの内、磁気コンパス データを出力するセンサがある場合、
　　data_rdy_status　|= DMP_Data_ready_Secondary_Compass(0x0008)
　　inv_event_control |= DMP_Motion_Event_Control_Compass_Calibr(0x0080)
A.8　スリープ解除
A.8.1　BANK0 を選択
A.8.2　AGB0_REG_PWR_MGMT_1(0x06)レジスタの SLEEP を 0 に設定
A.9　省電力モード解除
A.9.1　BANK0 を選択
A.9.2　AGB0_REG_PWR_MGMT_1(0x06)レジスタの LP_EN を 0 に設定
A.10　確度出力を確定
A.10.1 delta2(確度出力)をクリア
A.10.2 delta で 16-bit accel がオンになっている場合、
　　delta2 |= DMP_Data_Output_Control_2_Accel_Accuracy(0x4000)
A.10.3 delta で 32-bit calibrated gyro 又 16-bit gyro がオンになっている場合、
　　delta2 |= DMP_Data_Output_Control_2_Accel_Accuracy(0x4000)
A.10.4 delta で 32-bit calibrated compass 又は
　　16-bit compass 又は
　　32-bit 9-axis quaternion + heading accuracy 又は
　　32-bit Geomag rv + heading accuracy がオンの場合
　　delta2 |= DMP_Data_Output_Control_2_Compass_Accuracy
A.11　DMP メモリ.DATA_OUT_CTL1 に出力データ ビット パターンをライト
　　DATA_OUT_CTL1 : 4 * 16
　　dataOutCtrl1 = BYTE{ UInt8(delta >> 8), UInt8(delta & 0xff) }
　　(3)項 E の処理(DATA_OUT_CTL1, sizeof(dataOutCtrl1), dataOutCtrl1)
A.12　DMP メモリ.DATA_OUT_CTL2 に確度出力をライト(確度は header2 でデータ出力の有無が通知される)
　　DATA_OUT_CTL2 : 4 * 16 + 2
　　dataOutCtrl2 = BYTE{ UInt8(delta2 >> 8), UInt8(delta2 & 0xff) }
　　(3)項 E の処理(DATA_OUT_CTL2, sizeof(dataOutCtrl2), dataOutCtrl2)
A.13　DMP メモリ.DATA_RDY_STATUS にデータ レディ ステータスをライト
　　DATA_RDY_STATUS : 8 * 16 + 10
　　dataRdySt = BYTE{ data_rdy_status >> 8, data_rdy_status & 0xff }
　　(3)項 E の処理(DATA_RDY_STATUS, sizeof(dataRdySt), dataRdySt)
A.14　delta で 32-bit 9-axis quaternion + heading accuracy がオンになっている場合、
　　inv_event_control |= DMP_Motion_Event_Control_9axis(0x0040)
A.15　delta で DMP_Data_Output_Control_1_Step_Detector(0x0010)又は
　　DMP_Data_Output_Control_1_Step_Ind_0(0x0001)又は
　　DMP_Data_Output_Control_1_Step_Ind_1(0x0002)又は
　　DMP_Data_Output_Control_1_Step_Ind_2(0x0004)がオンになっている場合、
　　inv_event_control |= DMP_Motion_Event_Control_Pedometer_Interrupt(0x2000)
A.16　delta で DMP_Data_Output_Control_1_Geomag(0x0100)がオンになっている場合、
　　inv_event_control |= DMP_Motion_Event_Control_Geomag(0x0008)
A.17　DMP メモリ.MOTION_EVENT_CTL にイベント制御をライト
　　MOTION_EVENT_CTL : 4 * 16 + 14
　　motionEvCtrl = BYTE{ UInt8(inv_event_control >> 8), UInt8(inv_event_control & 0xff) }
　　(3)項 E の処理(MOTION_EVENT_CTL, sizeof(motionEvCtrl), motionEvCtrl)
A.18　省電力モード有効
A.18.1 BANK0 を選択
A.18.2 AGB0_REG_PWR_MGMT_1(0x06)レジスタの LP_EN を 1 に設定

B.　＜センサ データ出力レート設定(レジスタ アドレス, カウンタ アドレス, レート)＞

B.1　スリープ解除
B.1.1　BANK0 を選択
B.1.2　AGB0_REG_PWR_MGMT_1(0x06)レジスタの SLEEP を 0 に設定
B.2　省電力モード解除
B.2.1　BANK0 を選択
B.2.2　AGB0_REG_PWR_MGMT_1(0x06)レジスタの LP_EN を 0 に設定
B.3　DMP メモリ.レジスタ アドレスにレートをライト
rate = BYTE{ UInt8(レート >> 8), UInt8(レート & 0xff) }
(3)項 E の処理(レジスタ アドレス, sizeof(rate), rate)
B.4　DMP メモリ.カウンタ アドレスをクリア
counter = BYTE{ 0x00, 0x00 }
(3)項 E の処理(カウンタ アドレス, sizeof(counter), counter)
B.5　省電力モード有効
B.5.1　BANK0 を選択
B.5.2　AGB0_REG_PWR_MGMT_1(0x06)レジスタの LP_EN を 1 に設定

(5) ICM-20948 の DMP の演算結果を利用する場合のデータ読み出し処理

(各処理が正常に完了しない場合は、エラー ログを出力して処理を中止するようになっています。)

1　FIFO カウント取得(A の処理を参照)
2　FIFO カウントがヘッダ サイズ(2)に満たない時は「有効データなし」として処理を終了
3　ヘッダ(UInt16)を取得 (B の処理を参照)
4　ヘッダをエンディアン変換
5　ヘッダにより、"ヘッダ 2 あり"が指定されている場合、
　ヘッダ 2 を取得(B の処理を参照)
　ヘッダ 2 をエンディアン変換
6　ヘッダで"データあり"が指定されているデータを FIFO から取得(C の処理を参照)

ヘッダで指定されるデータ

ビット	名称	バイト数	エンディアン変換テーブル名
0x8000	16-bit accel	6	DMP_PQuat6_Byte_Ordering
0x4000	16-bit gyro	12	DMP_Raw_Gyro_Byte_Ordering
0x2000	16-bit compass	6	DMP_PQuat6_Byte_Ordering
0x1000	16-bit ALS	8	なし
0x0800	32-bit 6-axis quaternion	12	DMP_Quat6_Byte_Ordering
0x0400	32-bit 9-axis quaternion + heading accuracy	14	DMP_Quat9_Byte_Ordering
0x0200	16-bit pedometer quaternion	6	DMP_PQuat6_Byte_Ordering
0x0100	32-bit Geomag rotation vector + heading accuracy	14	DMP_Quat9_Byte_Ordering
0x0080	16-bit Pressure	6	なし
0x0040	32-bit calibrated gyro*1	0	
0x0020	32-bit calibrated compass	12	DMP_Quat6_Byte_Ordering
0x0010	Pedometer Step Detector	4	UInt32 型で変換
0x0008	Header2*2	2	UInt16 型で変換

(*1) 32-bit calibrated gyro(0x0040)のビットが 1 であっても、FIFO にデータは存在しない。
(*2) 5 の処理で FIFO から読み出される。

7　ヘッダ 2 で"データあり"が指定されているデータを FIFO から取得(B と C の処理を参照)

ヘッダ 2 で指定されるデータ

ビット	名称	バイト数	エンディアン変換テーブル名
0x4000	Accel Accuracy	2	UInt16 型で変換
0x2000	Gyro Accuracy	2	UInt16 型で変換
0x1000	Compass_Accuracy	2	UInt16 型で変換
0x0800	Fsync*1	0	
0x0400	Pickup	2	UInt16 型で変換
0x0080	Activity Recognition	6	DMP_Activity_Recognition_Byte_Ordering
0x0040	Secondary On Off	2	DMP_Secondary_On_Off_Byte_Ordering

(*1) Fsync(0x0800)のビットが 1 であっても、FIFO にデータは存在しない。

8　フッタ(UInt16)を FIFO から取得(B の処理を参照)
9　フッタをエンディアン変換
10　FIFO カウントが 0 より大きい場合は、"未読み出しのデータあり"で終了、そうでない場合は処理完了

A.　＜FIFO カウント取得＞

A.1　BANK0 を選択

A.2　H ← AGB0_REG_FIFO_COUNT_H(0x70) | 0x80 レジスタをリード

A.3　L ← AGB0_REG_FIFO_COUNT_H(0x71) | 0x80 レジスタをリード

A.4　((H << 8) | L)を FIFO カウント値とする

B.　＜FIFO リード(格納アドレス, リード サイズ, &FIFO カウント)＞

B.1　FIFO カウントがリード サイズに満たない場合、FIFO カウントを再取得(A の処理を参照)

B.2　FIFO カウントがリード サイズに満たない場合、「不完全データ検出」として全処理を終了

B.3　BANK0 を選択

B.4　AGB0_REG_FIFO_R_W(0x72)| 0x80 レジスタをリード サイズ分バースト リード

B.5　FIFO カウント -= リード サイズ

C.　＜FIFO データ リード(格納アドレス, エンディアン変換テーブル, リード サイズ, &FIFO カウント)＞

C.1　B の処理(格納アドレス, リード サイズ, FIFO カウント)

C.2　エンディアン変換テーブルが指定されている場合は、エンディアン変換を実施

(6) ICM-20948 の DMP の演算結果を利用する場合の磁気コンパス バイアス補正

(5)項までの処理を実装して、DMP 方向定位(INV_ICM20948_SENSOR_ORIENTATION)で動作させた結果、奇妙なドリフトが発生したので磁気コンパスのバイアス値を求めて、それを補正する処理を追加しました。(これは、サンプル コードにない処理なので、ドリフトを回避する正規の方法があるかも知れません。)

1　補正値を手動で求める

1.1　ICM20948 を水平に置き、DMP 未使用で起動し、Z 軸周りに一回転させた XY 地磁気データと、Y 軸周りに一回転させた XZ 地磁気データのログを保存(カンマ区切りの CSV 形式)

1.2　1.1 で保存した地磁気データをエクセルで開いて、分布グラフを作成

例 (調達した SparkFun 9DoF IMU)

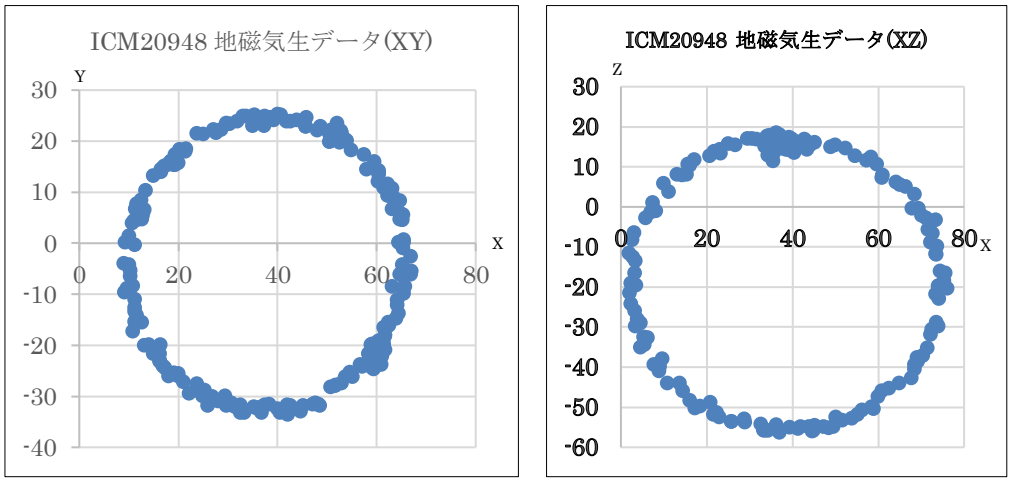


図 6-1 バイアス補正前地磁気データ表示例

1.3　図 6-1 の青の円の中心が(0,0)になるように X/Y/Z のオフセットを求める

例 (X 軸の補正=-38.0,Y 軸の補正=3.75,Z 軸の補正=19 で補正)

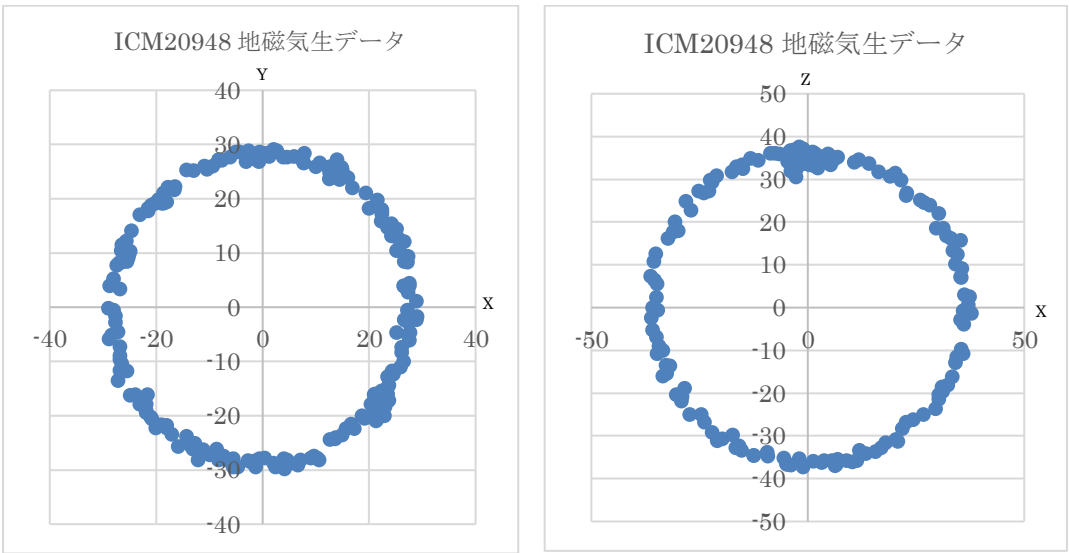


図 6-2 バイアス補正後の地磁気データ表示例

2　磁気コンパス バイアス値(バイアス補正値×-1.0)を DMP メモリに設定する関数を作成
磁気コンパス バイアス値設定処理(X 軸のバイアス値, Y 軸のバイアス値, Z 軸のバイアス値)
// 各バイアス値は倍精度(単位：μT)
2.1　各軸のずれに 1μT のスケール(2^{16}=65536)を掛けて整数化
2.2　整数化した各軸のずれをエンディアン変換
2.3　(3)項 E の処理でエンディアン変換した整数型の各軸のバイアス値を DMP メモリにライト
X 軸用補正値アドレス= CPASS_BIAS_X = (126 * 16 + 4)
Y 軸用補正値アドレス= CPASS_BIAS_Y = (126 * 16 + 8)
Z 軸用補正値アドレス= CPASS_BIAS_Z = (126 * 16 + 12)
3　プログラムの(3)項の処理の後（(4)項の処理を実行する前）に 2 で作成した磁気コンパス バイアス値設定処理の呼び出しを追加

7. データ確認用ツール

ICM-20948 から得られる姿勢データをビジュアルで確認できる CINEMA 4D 用のツール（簡単なタグプラグイン）を作成しましたので、本書の最後にソース コードを添付します。

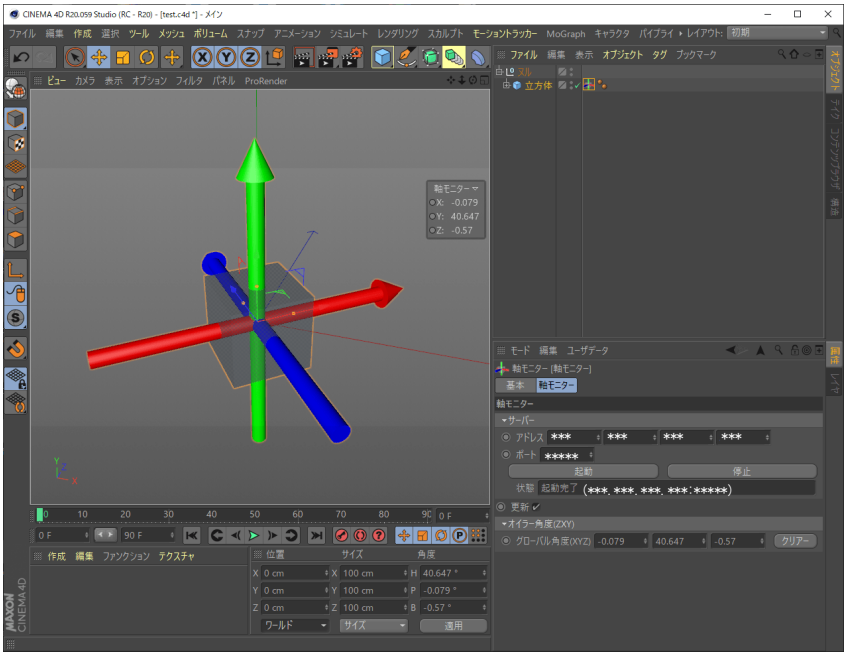


図 7-1 姿勢データの表示例

8. おわりに

TDK InvenSense ICM-20948 のデータ シート(英語)に設定手順の記載が見当たらなかったなかったので、長期戦を覚悟しましたが、SparkFun 9DoF IMU の Arduino 用のライブラリを参照できたので、短時間で動作させることが出来ました。(DMP を利用する場合、データ シートに記載されていないレジスタへのアクセスが必要です。)

ICM-20948 を利用するメリットがあるかどうかですが、間違いなく「ある」と思います。勿論、使用目的や動作環境によりますが、ロボットの基準部位の姿勢検出などに利用できると思います。(少し工夫が必要かも知れません。)

9. 添付資料

ICM-20948 姿勢データ　ビジュアル確認用 CINEMA 4D プラグイン　ソースコード

[TagMonAxes.h]

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// 軸モニター管理クラス
//

class TagMonAxes : public TagData
{
    // 構築
    TagMonAxes();
    // 破棄
    ~TagMonAxes();

public:
    // 軸モニター管理生成
    static NodeData* Alloc(void);

    // UIコンテナの初期化
    virtual Bool Init(GeListNode* node);
    // メッセージ ハンドラ
    virtual Bool Message(GeListNode* node, Int32 type, void* data);
    // 実行タイミング追加
    virtual Bool AddToExecution(BaseTag* tag, PriorityList* list);
    // 実行ハンドラ
    virtual EXECUTIONRESULT Execute(BaseTag* tag, BaseDocument* doc, BaseObject* op, BaseThread* bt, Int32 priority,
EXECUTIONFLAGS flags);

private:
    // UDPサーバ生成・破棄処理用スピン ロック
    static Spinlock smUdpSrvLock;
    // UDPサーバの参照カウンド
    static Int32 smRefCount;
    // UDPサーバの接続先軸モニター管理
    static TagMonAxes* volatile smConnectedMonAxes;
    // 最新の角度
    static Vector smLastEulerZXY;

    // 非同期IOサービス
    static AioServiceRef smAioService;
    // UDPサーバ
    static NetworkUdpServerRef smUdpServer;
    // UDPサーバ起動確認
    static Bool isUdpServerRunning(void) {
        geBreakRet(Bool(smUdpServer) == true, , false);
        return !smUdpServer.IsStopped();
    }
    // ドキュメント ロード ハンドラ (UDPサーバの復帰)
    void docLoaded(GeListNode* node, BaseDocument* doc);
    // コマンド実行
    void commandExecute(GeListNode* node, Int32 id, DescriptionCommand* command);
    // UDPサーバ起動
    void startUdpServer(BaseContainer* bc);
    // UDP受信コールバック
    static ResultVoid UDPCallback(ResultVoid result, AioBuffer receivedData, NetworkIpAddrPort sender);
    // UDPサーバ停止
    void stopUdpServer(BaseContainer* bc);
    // 最新の角度クリア
    void clearLastEulerZXY(BaseContainer* bc) {
        smLastEulerZXY.SetZero();
    }

    // 軸のグローバル角度設定
    void setMonGlobalRot(const Vector& globalRot);
    // 接続状況表示 (接続)
    void statConnected(BaseContainer* bc);
    // 接続状況表示 (切断)
    void statDisconnected(BaseContainer* bc);
};
```

[TagMonAxes.cpp]

```
//////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// 軸モニター管理クラス
//

// UDPサーバ生成・破棄処理用スピン ロック
Spinlock TagMonAxes::smUdpSrvLock;
// UDPサーバの参照カウント
Int32 TagMonAxes::smRefCount = 0;
// UDPサーバの接続先軸モニター管理
TagMonAxes* volatile TagMonAxes::smConnectedMonAxes = nullptr;
// 最新のグローバル角度
Vector TagMonAxes::smLastEulerZXY;

// 非同期IOサービス
AioServiceRef TagMonAxes::smAioService;
// UDPサーバ
NetworkUdpServerRef TagMonAxes::smUdpServer;

//////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// 機能　　:　　軸モニター管理生成
// 戻り値　:　　軸モニター管理
//

NodeData* TagMonAxes::Alloc(void)
{
    return NewObjClear(TagMonAxes);
}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// 機能　　:　　軸モニター管理構築
//

TagMonAxes::TagMonAxes()
{
    smUdpSrvLock.Lock();
    ++smRefCount;
    smUdpSrvLock.Unlock();
}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// 機能　　:　　軸モニター管理破棄
//

TagMonAxes::~TagMonAxes()
{
    smUdpSrvLock.Lock();
    geBlock(smConnectedMonAxes == this, smConnectedMonAxes = nullptr);
    if (--smRefCount <= 0)
    {
        geBlock(Bool(smUdpServer) == true, smUdpServer.Stop(); smUdpServer = NetworkUdpServerRef());
        geBlock(Bool(smAioService) == true, smAioService.StopAndWait(); smAioService = AioServiceRef());
        smRefCount = 0;
    }
    smUdpSrvLock.Unlock();
}

//////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// 機能　　:　　UIコンテナの初期化
// 戻り値　:　　処理結果
//　　　　:　　　　　　true　　: 正常終了
//　　　　:　　　　　　false　　: 異常終了
//

Bool TagMonAxes::Init(GeListNode* node)
{
    // コンテナ データを初期化
```

```
    BaseContainer* bc = static_cast<BaseTag*>(node)->GetDataInstance();

    bc->SetInt32(DID_TMONAXES_SVR_ADDR1, ***);
    bc->SetInt32(DID_TMONAXES_SVR_ADDR2, ***);
    bc->SetInt32(DID_TMONAXES_SVR_ADDR3, ***);
    bc->SetInt32(DID_TMONAXES_SVR_ADDR4, ***);
    bc->SetInt32(DID_TMONAXES_SVR_PORT, *****);
    bc->SetString(DID_TMONAXES_SVR_STAT, GeLoadString(IDS_TAG_MONAXES_STOPED));
    bc->SetBool(DID_TMONAXES_UPDATE, true);
    bc->SetBool(DID_TMONAXES_SRV_STARTED, false);

    return true;
}

////////////////////////////////////////////////////////////////////////////////////////////////////
//
// 機能      :            メッセージ ハンドラ
// 戻り値    :            処理結果
//           :                        true        : 正常終了
//           :                        false       : 異常終了
//

Bool TagMonAxes::Message(GeListNode* node, Int32 type, void* t_data)
{
    Bool sts = true;

    switch (type)
    {
    case MSG_DOCUMENTINFO:
        {
            DocumentInfoData* docInfo = reinterpret_cast<DocumentInfoData*>(t_data);
            if (docInfo->type == MSG_DOCUMENTINFO_TYPE_LOAD)
            {
                docLoaded(node, docInfo->doc);
            }
            break;
        }

    case MSG_DESCRIPTION_COMMAND:
        {
            DescriptionCommand* command = reinterpret_cast<DescriptionCommand*>(t_data);
            Int32 id = command->_descId[0].id;
            commandExecute(node, id, command);
            break;
        }

    default:
        break;
    }

    return sts;
}

////////////////////////////////////////////////////////////////////////////////////////////////////
//
// 機能      :            ドキュメント ロード ハンドラ (UDPサーバの復帰)
//

void TagMonAxes::docLoaded(GeListNode* node, BaseDocument* doc)
{
    BaseContainer* bc = static_cast<BaseTag*>(node)->GetDataInstance();

    // サーバを起動した状態で保存されたドキュメントがロードされた場合は、サーバを起動する
    if (bc->GetBool(DID_TMONAXES_SRV_STARTED) == true)
    {
        startUdpServer(bc);
    }

    // 最新の角度を復帰
    smLastEulerZXY = bc->GetVector(DID_TMONAXES_GLOBAL_ROT);
}

////////////////////////////////////////////////////////////////////////////////////////////////////
//
// 機能      :            コマンド実行
//

void TagMonAxes::commandExecute(GeListNode* node, Int32 id, DescriptionCommand* command)
{
    switch (id)
    {
    case DID_TMONAXES_SVR_START:
        startUdpServer(static_cast<BaseTag*>(node)->GetDataInstance());
        break;

    case DID_TMONAXES_SVR_STOP:
        stopUdpServer(static_cast<BaseTag*>(node)->GetDataInstance());
        break;

    case DID_TMONAXES_GROT_CLEAR:
        clearLastEulerZXY(static_cast<BaseTag*>(node)->GetDataInstance());
    default:
        break;
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////
//
// 機能      :            UDPサーバ起動
//

void TagMonAxes::startUdpServer(BaseContainer* bc)
{
    if (Bool(smAioService) == false)
    {
        // 非同期IOサービスを生成
        ResultMemT(AioServiceRef) stCreAio = AioServiceRef::Create();
        geBlockElse(stCreAio == RESULT_OK,
            smAioService = stCreAio.GetValue(),
            MessageDialog(IDS_TAG_MONAXES_START_ERR_AIO); return);
    }
    // 非同期IOサービスを起動
    smAioService.Start();

    if (Bool(smUdpServer) == false)
    {
        // 自局のIPアドレスとポート番号を確定
        Int32 addr1 = bc->GetInt32(DID_TMONAXES_SVR_ADDR1);
        Int32 addr2 = bc->GetInt32(DID_TMONAXES_SVR_ADDR2);
        Int32 addr3 = bc->GetInt32(DID_TMONAXES_SVR_ADDR3);
        Int32 addr4 = bc->GetInt32(DID_TMONAXES_SVR_ADDR4);
        Int32 port = bc->GetInt32(DID_TMONAXES_SVR_PORT);
        NetworkIpAddrPort addrPort =
            NetworkIpAddrPort(UChar(addr1), UChar(addr2), UChar(addr3), UChar(addr4), port);

        // UDPサーバを生成
        ResultT(NetworkUdpServerRef) stCreUdpSrv =
            NetworkUdpInterface::CreateUdpServer(addrPort, UDPCallback, smAioService);
        geBlockElse(stCreUdpSrv == RESULT_OK,
            smUdpServer = stCreUdpSrv.GetValue(),
            MessageDialog(IDS_TAG_MONAXES_START_ERR_ADDR); return);
    }
    // UDPサーバを起動
    smUdpServer.Start();
}

////////////////////////////////////////////////////////////////////////////////////////////////////
//
// 機能      :            UDP受信コールバック
//

ResultVoid TagMonAxes::UDPCallback(ResultVoid result, AioBuffer receivedData, NetworkIpAddrPort sender)
{
    if (smConnectedMonAxes != nullptr)
    {
        if (receivedData.GetCount() == sizeof(Vector32))
        {
            const Vector32& xyzDeg = *reinterpret_cast<const Vector32*>(receivedData.GetFirst());
            smConnectedMonAxes->setMonGlobalRot(Vector(xyzDeg.x, xyzDeg.y, xyzDeg.z));
        }
    }
    return RESULT_OK;
```

```
}

//////////////////////////////////////////////////////////////////////////////////////////////////
//
// 機能      :            UDPサーバ停止
//

void TagMonAxes::stopUdpServer(BaseContainer* bc)
{
    if (smConnectedMonAxes == this)
    {
        // UDPサーバを停止
        geBlock(Bool(smUdpServer) == true, smUdpServer.Stop());
        // 非同期サービスを停止
        geBlock(Bool(smAioService) == true, smAioService.StopAndWait());
    }
}

//////////////////////////////////////////////////////////////////////////////////////////////////
//
// 機能      :            軸のグローバル角度設定
// 備考      :            角度設定はHPB固定

void TagMonAxes::setMonGlobalRot(const Vector& globalRot)
{
    BaseTag* tag = static_cast<BaseTag*>(Get());
    BaseContainer* bc = tag->GetDataInstance();

    if (bc->GetBool(DID_TMONAXES_UPDATE) == true)
    {
        BaseObject* op = tag->GetObject();
        Vector radRot(DegToRad(globalRot.x), DegToRad(globalRot.y), DegToRad(globalRot.z));
        Matrix mg(op->GetMg().off, HPBToMatrix(radRot, ROTATIONORDER::ZXYGLOBAL).sqmat);
        op->SetMg(mg);

        bc->SetVector(DID_TMONAXES_GLOBAL_ROT, globalRot);
        smLastEulerZXY = globalRot;

        op->Message(MSG_CHANGE);
        EventAdd();
    }
}

//////////////////////////////////////////////////////////////////////////////////////////////////
//
// 機能      :            接続状況表示（接続）
//

void TagMonAxes::statConnected(BaseContainer* bc)
{
    // UDPサーバの接続先軸モニター管理を設定
    smConnectedMonAxes = this;
    // サーバ起動状態を更新
    bc->SetBool(DID_TMONAXES_SRV_STARTED, true);
    // 「サーバ起動完了」を表示
    NetworkIpAddrPort addrPort = smUdpServer.GetLocalAddress();
    U8String srvStat = U8String(GeLoadString(IDS_TAG_MONAXES_STARTED)) + u8"(" + addrPort.ToString(nullptr) + u8")";
    bc->SetString(DID_TMONAXES_SVR_STAT, srvStat);
}

//////////////////////////////////////////////////////////////////////////////////////////////////
//
// 機能      :            接続状況表示（切断）
//

void TagMonAxes::statDisconnected(BaseContainer* bc)
{
    // UDPサーバと軸モニター管理の接続を切断
    smConnectedMonAxes = nullptr;
    // サーバ起動状態を更新
    bc->SetBool(DID_TMONAXES_SRV_STARTED, false);
    // 「停止」を表示
    bc->SetString(DID_TMONAXES_SVR_STAT, GeLoadString(IDS_TAG_MONAXES_STOPED));
}

//////////////////////////////////////////////////////////////////////////////////////////////////
//
// 機能      :            実行タイミング追加
// 戻り値    :            処理結果
//                                true        : 正常終了
//                                false       : 異常終了
// 備考      :            なし
//

Bool TagMonAxes::AddToExecution(BaseTag* tag, PriorityList* list)
{
    list->Add(tag, EXECUTIONPRIORITY_INITIAL, EXECUTIONFLAGS::NONE);

    return true;
}

//////////////////////////////////////////////////////////////////////////////////////////////////
//
// 機能      :            実行ハンドラ
// 戻り値    :            処理結果
//                                EXECUTIONRESULT_OK                                  : 正常終了
// 備考      :            なし
//

EXECUTIONRESULT TagMonAxes::Execute(BaseTag* tag, BaseDocument* doc, BaseObject* op, BaseThread* bt, Int32
priority, EXECUTIONFLAGS flags)
{
    if (N(flags & EXECUTIONFLAGS::RENDER) == 0)
    {
        BaseContainer* bc = tag->GetDataInstance();
        geBlockElse(isUdpServerRunning() == true, statConnected(bc), statDisconnected(bc));
        bc->SetVector(DID_TMONAXES_GLOBAL_ROT, smLastEulerZXY);
    }

    return EXECUTIONRESULT::OK;
}
```

```
[TagMonAxes.res]
CONTAINER TmonAxes
{
    NAME TmonAxes;
    INCLUDE Texpression;

    GROUP ID_TMONAXES_PROPERTIES
    {   DEFAULT 1;
        GROUP ID_TMONAXES_SERVER
        {   DEFAULT 1;
            GROUP
            {   COLUMNS 4;
                LONG DID_TMONAXES_SVR_ADDR1                 { MIN 0; MAX 255; }
                LONG DID_TMONAXES_SVR_ADDR2                 { MIN 0; MAX 255; }
                LONG DID_TMONAXES_SVR_ADDR3                 { MIN 0; MAX 255; }
                LONG DID_TMONAXES_SVR_ADDR4                 { MIN 0; MAX 255; }
            }
            LONG DID_TMONAXES_SVR_PORT                      { MIN 49152; MAX 65535; }
            GROUP
            {   COLUMNS 2;
                BUTTON DID_TMONAXES_SVR_START           { FIT_H; SCALE_H; }
                BUTTON DID_TMONAXES_SVR_STOP            { FIT_H; SCALE_H; }
            }
            STRING DID_TMONAXES_SVR_STAT                { ANIM OFF; }
        }
        BOOL DID_TMONAXES_UPDATE                                        { }
        BOOL DID_TMONAXES_SRV_STARTED                   { HIDDEN; }
        GROUP ID_TMONAXES_EULER_ZXY
        {   DEFAULT 1;
            GROUP
            {   COLUMNS 2;
                VECTOR DID_TMONAXES_GLOBAL_ROT                  { }
                BUTTON DID_TMONAXES_GROT_CLEAR                  { FIT_H; SCALE_H; }
            }
        }
    }
}
```

改定履歴	改　　定　　内　　容	検認	照査	作成
初期作成 21/8/12		―	―	市来 博記
B 21/12/15	5 準備の SPI 通信の準備 6 項のソース コードの例を修正した。 ioctl(fd, SPI_IOC_RD_～, 格納先領域)行の格納先領域の初期値を 0 に修正した。 struct spi_ioc_transfer tr の初期化に cs_change の設定行の記載を追加した。 7 項のソース コードの例の不要なコメント行を削除した。	―	―	市来 博記
C 21/12/25	文書ファイルのプロパティを設定した。	―	―	市来 博記