OMRON

MEMS差圧センサ

形D6F-PH

ユーザーズマニュアル

MEMS差圧センサ

CDSC-025B

目次

1　概要

本ユーザーズマニュアルは、弊社 MEMS 差圧センサ（形 D6F-PH）のご使用方法、特記事項などを示すものです。なお本資料は製品カタログを補足するものであり、実際のご使用にあたっては製品カタログもあわせてご使用下さい。

2　構造

図 1 に MEMS 差圧センサ（D6F-PH）の内部断面イメージ図を示します。圧力導入孔から入った空気の流れは、フローセンサチップ表面を通り、反対側の導入孔から出て行きます。この時に MEMS フローセンサの動作原理に基づき気体の流れを検出します。

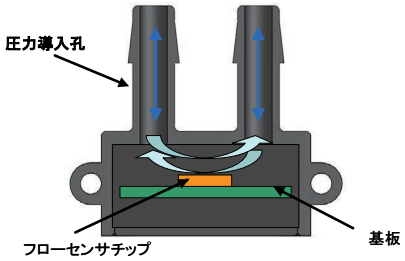
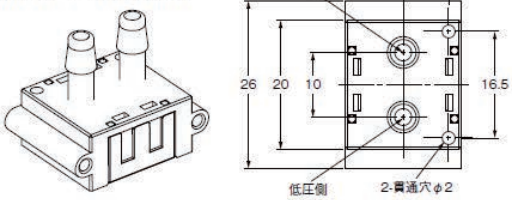
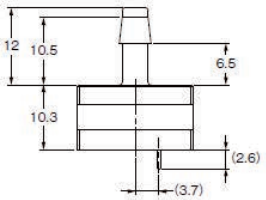


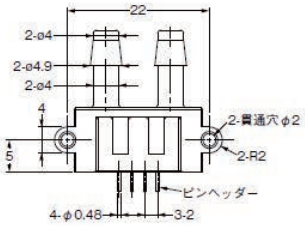
図 1 MEMS 差圧センサ（D6F-PH）の内部断面イメージ図

3　外形寸法図

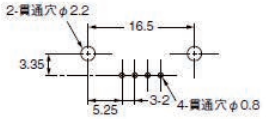
基板実装タイプ
形D6F-PH0025AD1
形D6F-PH0505AD3
形D6F-PH5050AD3

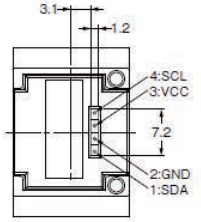






ピンヘッダー取り付け推奨サイズ
（許容差：±0.1）





チューブ

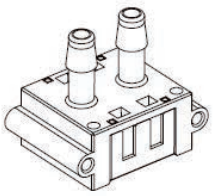
ゴム・ウレタン・ナイロン製などのチューブで押し込んで抜けないように取り付けてください。
ウレタン系チューブであれば外径φ6mm、内径φ4mmのものを推奨します。

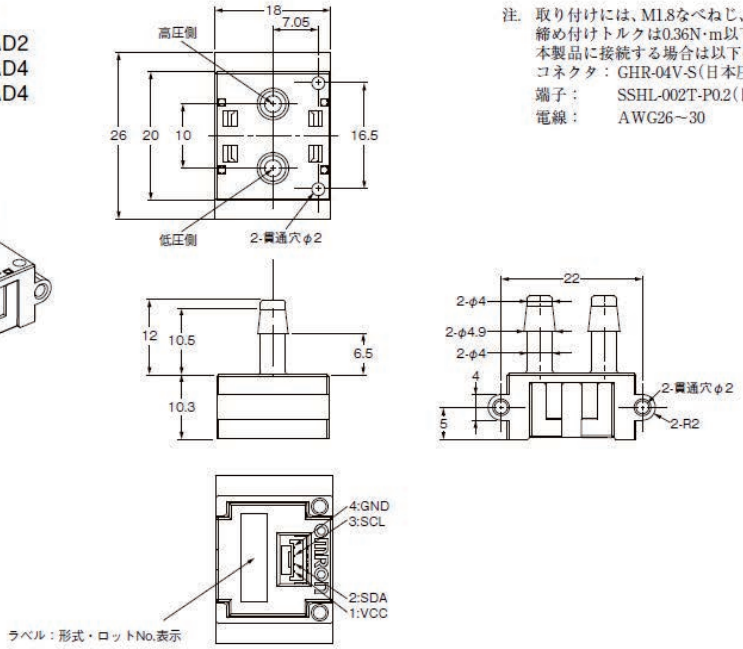
はんだ条件

はんだごてを使用し、押圧100gf以下、温度350℃、時間5秒としてください。

図 2 D6F-PH0505AD3 / D6F-PH0025AD1 / D6F-PH5050AD3 の外形寸法図

コネクタタイプ
形D6F-PH0025AD2
形D6F-PH0505AD4
形D6F-PH5050AD4



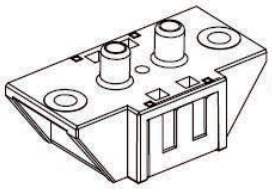


注. 取り付けには、M1.8なべねじ、もしくはタッピングねじを使用し、締め付けトルクは0.36N·m以下とすること。
本製品に接続する場合は以下のコネクタを使用すること。
コネクタ：GHR-04V-S(日本圧着端子製造(株)製)
端子：SSHL-002T-P0.2(日本圧着端子製造(株)製)
電線：AWG26～30

図 3 D6F-PH0505AD4 / D6F-PH0025AD2 / D6F-PH5050AD4 の外形寸法図

形D6F-PH0025AMD2
形D6F-PH0505AMD4
形D6F-PH5050AMD4

注1. 取付ネジはM3丸小ネジもしくはタッピングネジを使用し、締付けトルク1.0N・m以下とする。
注2. ネジ頭部およびワッシャー外径は6mm以下とする。
注3. 導入ポートのシール部は、φ7.0の溝で、P4Oリングとする。(JIS B 2401準拠)
注4. 本製品に接続する場合は以下のコネクタを使用すること。
コネクタ：GHR-04V-S(日本圧着端子製造(株)製)
端子：SSHL-002T-P0.2(日本圧着端子製造(株)製)
電線：AWG26～30



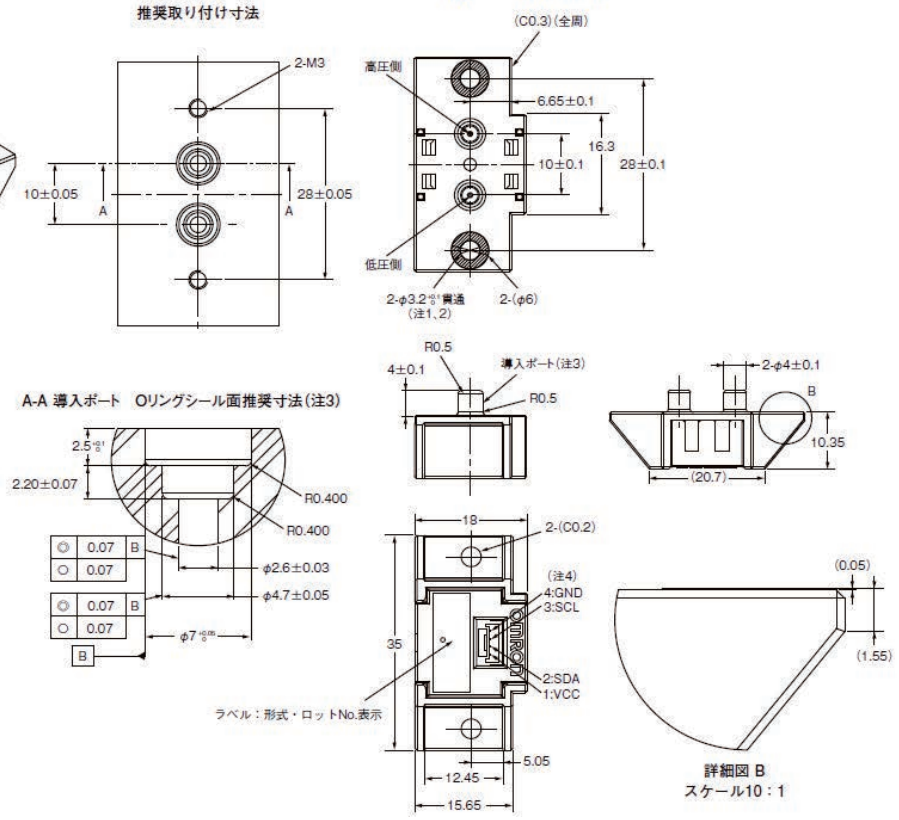


図 4 D6F-PH0505AMD4 / D6F-PH0025AMD2 / D6F-PH5050AMD4 の外形寸法図

4　動作原理

MEMS 差圧センサ（D6F-PH）は、主流路に設けたオリフィスにより分流した流れを熱式質量流量フローセンサで計測し、微差圧を検知します。

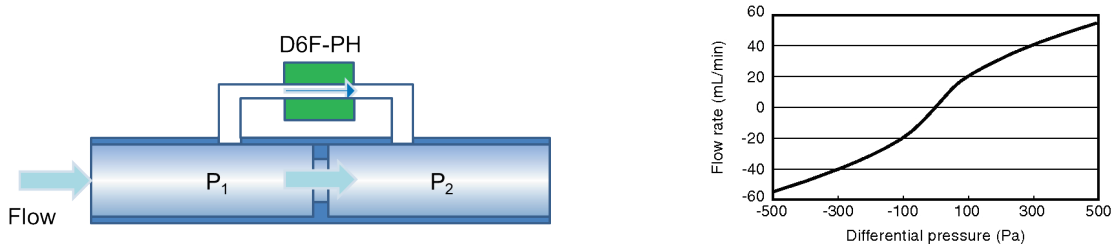
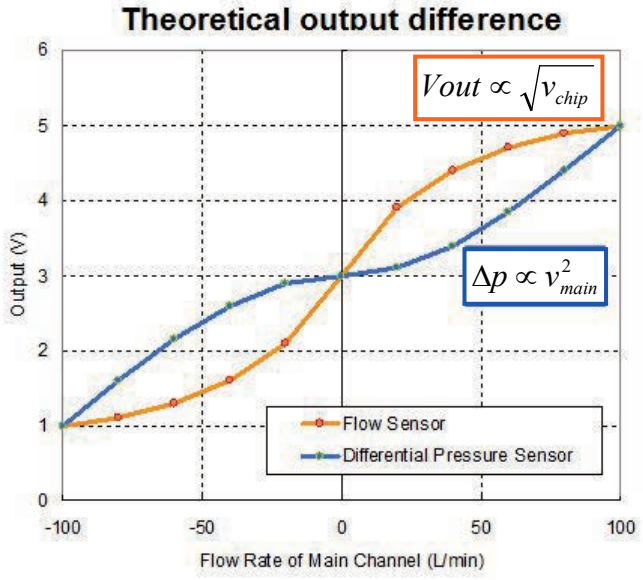


図 5　差圧センサの原理(a) と流量と差圧の関係(b)

5　製品の特長

MEMS 差圧センサ（D6F-PH）は熱式質量流量方式（熱フロー式）を用いる事で、メンブレン式差圧センサより低圧力域の変化をより高感度で検出することができます。



熱フロー式差圧センサ
　出力がフローセンサチップ表面を流れる気体流速の平方根に比例します。

メンブレン式差圧センサ
　差圧が主流路を流れる気体流速の2乗に比例します。

橙：熱フロー式差圧センサ
青：メンブレン式差圧センサ

図 6 メンブレン式差圧センサとの熱フロー式差圧センサの比較

表 1　D6F-PH ラインナップ

差圧範囲	継ぎ手	接続	形式
0-250Pa	タケノコ継ぎ手	基板実装	D6F-PH0025AD1
		コネクタ	D6F-PH0025AD2
	マニフォールド	コネクタ	D6F-PH0025AMD2
±50Pa	タケノコ継ぎ手	基板実装	D6F-PH0505AD3
		コネクタ	D6F-PH0505AD4
	マニフォールド	コネクタ	D6F-PH0505AMD4
±500Pa	タケノコ継ぎ手	基板実装	D6F-PH5050AD3
		コネクタ	D6F-PH5050AD4
	マニフォールド	コネクタ	D6F-PH5050AMD4

表 2　D6F-PH□□□□の主な仕様

項目	内容				
	Min	Typ	Max	単位	備考
差圧範囲	-50	-	50	Pa	D6F-PH0505AD3-□ D6F-PH0505AD4 D6F-PH0505AMD4
	0	-	250	Pa	D6F-PH0025AD1-□ D6F-PH0025AD2 D6F-PH0025AMD2
	-500	-	500	Pa	D6F-PH5050AD3-□ D6F-PH5050AD4 D6F-PH5050AMD4
分解能	-	12	-	bit	
ゼロ点精度（注）	-0.2	-	+0.2	Pa	
スパン精度（注）	-3	-	+3	%R.D.	
温度変動によるスパンシフト	-0.5	-	+0.5	%R.D.	10℃の変化に対して
応答時間	-	33	50	msec	12bit 分解能
使用周囲温度範囲	-20	-	80	℃	結露、氷結しないこと
保存周囲温度範囲	-40	-	80	℃	結露、氷結しないこと
使用周囲湿度範囲	35	-	85	%RH	結露、氷結しないこと
保存周囲湿度範囲	35	-	85	%RH	結露、氷結しないこと
電源電圧	2.3	3.3	3.6	VDC	
消費電流	-	-	6	mA	Vcc=3.3V、25℃
SCL 周波数	-	-	400	kHz	FAST Mode 対応

（注）　ゼロ点精度とスパン精度は独立した誤差であり、同時に満足するものではありません。

6　使用方法

6.1　流路への接続方法

微小圧力変化の測定対象となる主流路にオリフィスを設けることで、オリフィス前後に微小圧力差を発生させます。オリフィス前後に設けた圧力ポートからバイパス流路を D6F-PH に接続して下さい。この接続方式により、オリフィス前後に発生する微小圧力差を検出します。

(1) タケノコ継手の場合（D6F-PH0505AD3 / D6F-PH0025AD1 / D6F-PH5050AD3 / D6F-PH0505AD4 / D6F-PH0025AD2 / D6F-PH5050AD4）

D6F-PH との接続に使用するチューブは内径 4.0[mm]、主流路からセンサ間のバイパス流路長は 800[mm]以下として下さい。チューブは可能な限りまっすぐにしてください。

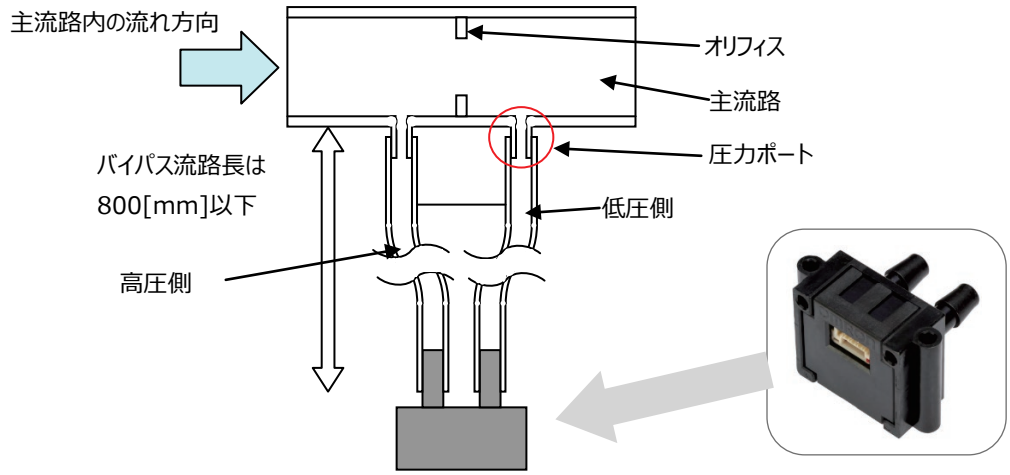


図 7 D6F-PH（タケノコ継手）の主流路への接続

(2) マニフォールドの場合（D6F-PH0505AMD4 / D6F-PH0025AMD2 / D6F-PH5050AMD4）

D6F-PH と接続する部分は O リング等でシールして取り付けてください。

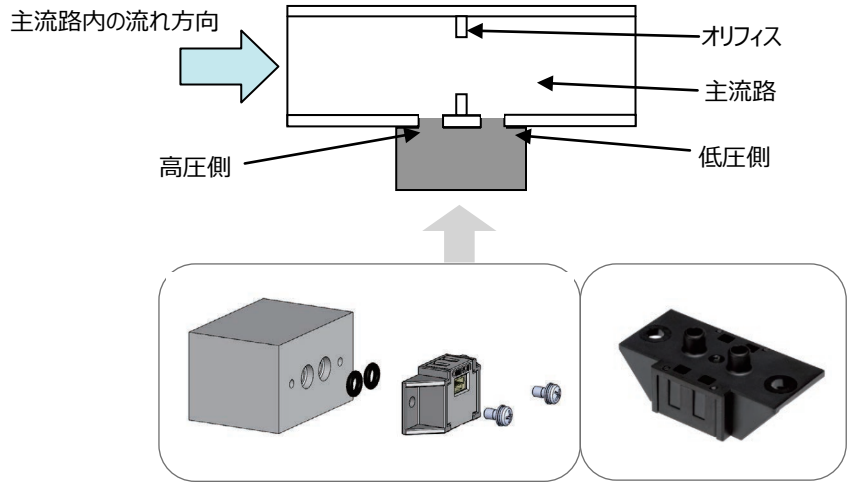


図 8 D6F-PH（マニフォールド継手）の主流路への接続

6.2 流路設計

D6F-PH に接続する流路は下記の手順で設計・評価してください。以下に示される数値は理論値であり、実際にはお客様のシステムで評価頂いた上で決定してください。

[STEP1]

下記の必要要件を決定してください。

・システムにおける主流路に流れる最大流量　：　Fmax
・圧力範囲（バイパス部分で許容できる圧損）　：　Pmax
・主流路の内径　：　D

Fmax はご使用のポンプの性能に依存します。Fmax の時にバイパスにかかる圧損が Pmax になります。

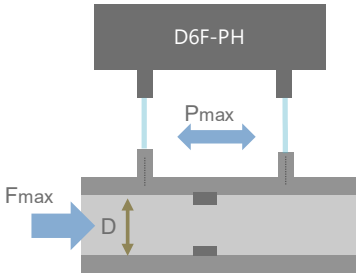


図 9 バイパス構造

[STEP2]

圧力範囲 Pmax で D6F-PH の型式が決まります。D6F-PH を選定してください。

- 0~250 Pa の場合　D6F-PH0025AD1 / D6F-PH0025AD2 / D6F-PH0025AMD2
- -50~50 Pa の場合　D6F-PH0505AD3 / D6F-PH0505AD4 / D6F-PH0505AMD4
- -500~500 Pa の場合　D6F-PH5050AD3 / D6F-PH5050AD4 / D6F-PH5050AMD4

[STEP3]

Fmax と D から、下記の表に従って、オリフィス径 d (mm)を決定してください。

表 3 オリフィス径 d (D6F-PH0025AD1 / D6F-PH0025AD2 / D6F-PH0025AMD2)

Flow rate:	(L/min)	10	20	30	50	100	150
Fmax	(m3/h)	0.6	1.2	1.8	3.0	6.0	9.0
D (mm)	10	3.61	5.04	6.05	7.40	8.92	9.44
	20	3.62	5.12	6.26	8.04	11.16	13.28
	30	3.62	5.12	6.27	8.09	11.39	13.86
	40	3.62	5.12	6.27	8.09	11.43	13.97
	50	3.62	5.12	6.27	8.09	11.44	14.00

表 4 オリフィス径 d (D6F-PH0505AD3 / D6F-PH0505AD4 / D6F-PH0505AMD4)

Flow rate:	(L/min)	5	10	20	30	40	50
Fmax	(m3/h)	0.6	1.2	1.8	3.0	6.0	9.0
D (mm)	10	3.81	5.30	7.11	8.13	8.72	9.09
	20	3.83	5.41	7.62	9.27	10.61	11.73
	30	3.83	5.41	7.65	9.36	10.78	12.03
	40	3.83	5.41	7.65	9.37	10.81	12.08
	50	3.83	5.41	7.66	9.37	10.82	12.10

表 5 オリフィス径 d (D6F-PH5050AD3 / D6F-PH5050AD4 / D6F-PH5050AMD4)

Flow rate:	(L/min)	10	20	30	50	100	150
Fmax	(m3/h)	0.6	1.2	1.8	3.0	6.0	9.0
D (mm)	10	3.04	4.27	5.18	6.48	8.25	9.01
	20	3.04	4.30	5.27	6.79	9.50	11.46
	30	3.04	4.31	5.27	6.80	9.60	11.72
	40	3.04	4.31	5.27	6.81	9.62	11.77
	50	3.04	4.31	5.27	6.81	9.62	11.78

例）主流路に流れる最大流量が 100 l/min になるように
ポンプを制御。その時にバイパス部分が 500 Pa の
圧損とするには、主流路の内径 D が 30mm の時、
オリフィス径 d は 9.6 ㎜。(表 4 参照)

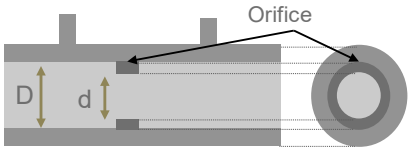


図 10 オリフィス

[STEP4]

STEP 3 で決定したオリフィス径 d を利用して、流路を設計・作製してください。

a = 5 mm

b = 5 mm

c = 1～2mm 程度

L = 10 D 以上

（助走距離 L は、まっすぐになっておくべき距離です。十分に助走距離が取れない場合は、L は左右対称の値にしておいてください。）

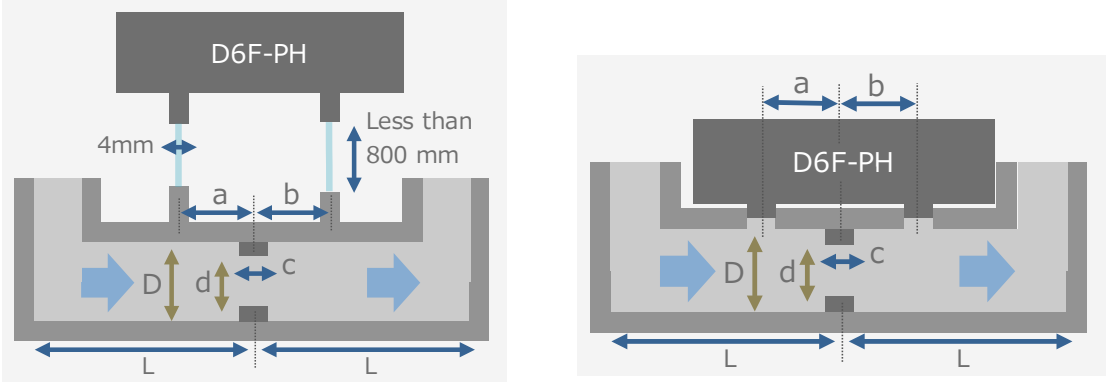


図 11 流路構造例 （左：タケノコ継手、 右：マニフォールド継手）

[STEP5]

設計・作製した流路を使って実測し、D6F-PH が正しい圧力値を出力しているか確認してください。

正しい値が出力されない場合は、流路設計の寸法値を変更し、正しい値になるよう調整してください。

（例）

流量 Fmax の時の D6F-PH の出力値を確認。

（理想は、流量が Fmax の時に、D6F-PH の圧力値は Pmax 付近を出力。）

D6F-PH の圧力値が、お客様のシステムで許容できないばらつきであれば、オリフィス径 d と主流路内径 D の比率の見直し実施。

・D6F-PH の出力が Pmax より大きい場合は、d/D の値を大きくする。

・D6F-PH の出力が Pmax より小さい場合は、d/D の値を小さくする。

新たに流路を作製し、再評価を実施する。

6.3 D6F-PH の電気的接続方法

D6F-PH は I2C 出力の為、データライン（SDA）とクロックライン（SCL）の各々にプルアップ抵抗が必要となります。図 6 に示すように Vcc との間にプルアップ抵抗 2.2[kΩ]（推奨値）を実装して下さい。
なお、お使い頂く配線長等に応じて、プルアップ抵抗と SCL の転送速度を適切な値に変更してお使い下さい。

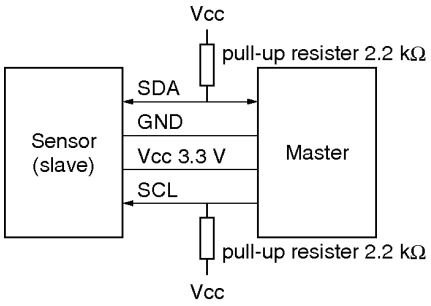


図 12 D6F-PH の電気的接続方法

※ フローセンサ接続に関してのご注意点
お客様でお使い頂く環境のノイズ影響により通信時にエラーが発生する可能性があります。その際には以下の点をご確認頂き、通信エラーを改善して頂く必要があります。

① 通信速度の確認
本製品はSCL 周波数が400 kHz まで対応しておりますが、通信エラーが発生しやすい場合は、SCL 周波数を低い周波数に設定してご使用されることをお勧め致します。

② 配線ケーブルの確認
お客様制御マイコンと弊社フローセンサの間を接続するケーブル長が長い場合、よりノイズの影響を受けやすくなります。この場合はシールドケーブルのご使用をお勧め致します。

③ プルアップ抵抗値の確認
本製品のI2C 通信には、プルアップ抵抗が必要となります。推奨抵抗値は2.2 [kΩ]としておりますが、お客様側制御マイコンと弊社フローセンサの間の接続ケーブル長に合わせて最適な抵抗値を選択して下さい。なお、センサ側からACK が返答されなければ、通信異常と判断して下さい。ACK 返答時間は、SCL の1 clock サイクル分になりますので、それ以上の時間ACK 返答がない場合には通信異常となります。その場合は、一度電源を落としていただく必要があります。

7　通信仕様

7.1 I2C インタフェース概要

表 6.　I2C 通信に関する基本仕様

型式		形 D6F-PH0025AD1 形 D6F-PH0505AD3 形 D6F-PH5050AD3 形 D6F-PH0025AD2 形 D6F-PH0505AD4 形 D6F-PH5050AD4 形 D6F-PH0025AMD2 形 D6F-PH0505AMD4 形 D6F-PH5050AMD4	形 D6F-PH0025AD1-1 形 D6F-PH0505AD3-1 形 D6F-PH5050AD3-1	形 D6F-PH0025AD1-2 形 D6F-PH0505AD3-2 形 D6F-PH5050AD3-2	形 D6F-PH0025AD1-3 形 D6F-PH0505AD3-3 形 D6F-PH5050AD3-3
通信方式		I2C			
スレーブアドレス	HEX	0x6C	0x6D	0x6E	0x6F
	BIN(7bit)	110_1100	110_1101	110_1110	110_1111
通信周波数		Max. 400kHz			
シグナル	SCL	Serial Clock			
	SDA	Data Signal			

表 7. I2C スレーブアドレスは以下のように表現されます。（0x6C の例）

Bit	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
								R/W
値	1	1	0	1	1	0	0	1/0

Write 時：スレーブアドレスのLSB を”0”にセットし、 D8h (1101_1000b)とする。

Read 時：スレーブアドレスの LSB を”1”にセットし、 D9h (1101_1001b)とする。

7.2 動作手順

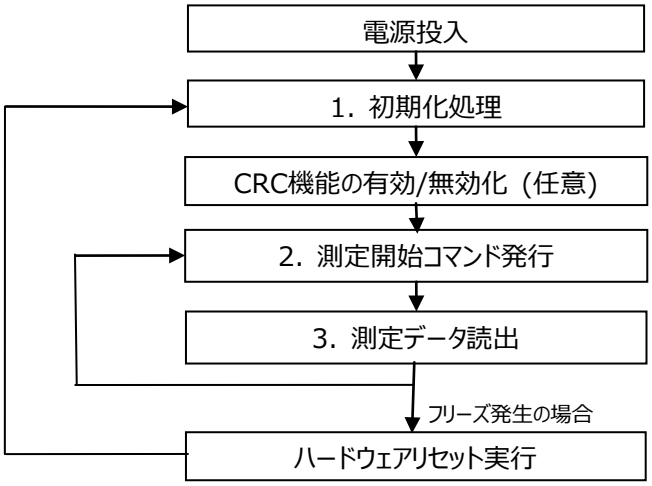


図 13 センサ動作時のフローチャート

・通信タイミングについて

項目	記号	備考欄
レスポンスタイム	α	α ≧ 33 ms
サンプリング間隔	β	β > α

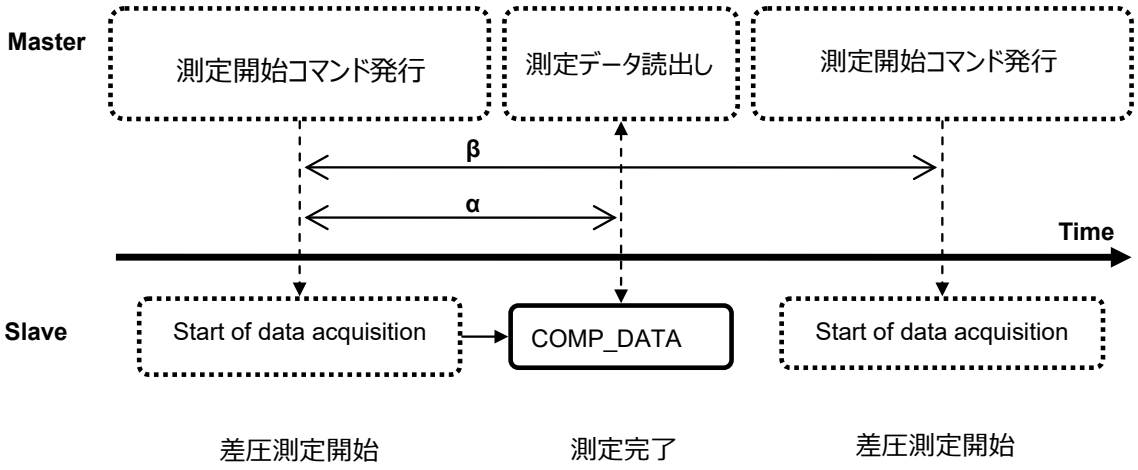


図 14 差圧測定の時間軸イメージ図

1.初期化処理

電源投入後200μs経過後に実施してください。

I2C コマンド:初期化レジスタ(0Bh)に 00h をライト。

START	Slave Address	ACK	Access Address	ACK	Write Data	ACK	STOP
S	D8h (6Ch (7b)+ 0)	A	0Bh	A	00h	A	P

2.測定開始コマンド発行

各種設定と共にMCUモードが実行(差圧測定実施)されます。書込後にこのレジスタを読出す事で、MCUで選択されたMUXの状態を読出す事ができます。処理実行後、MSビットが"0"にセットされます。MCU動作中はデバイスにアクセスしないで下さい。33msec経過後にアクセスしてください。

I2C コマンド: アクセスアドレスレジスタ(00h)を介してセンサ制御レジスタ（D040h）に06h書込 (MS=1 & MCU_on)

START	Slave Address	ACK	Access Address	ACK
S	D8h (6Ch (7b)+ 0)	A	00h	A

Reg Address H	ACK	Reg Address L	ACK	Serial Ctrl	ACK	Write Data	ACK	STOP
D0h	A	40h	A	18h	A	06h	A	P

3.測定データ読出

初期化処理後、最初のデータは正常なデータではないので、無視してください。

I2C コマンド: アクセスアドレスレジスタ(00h)を介して補正流量値データレジスタ（D051h、D052h）の読出指令2Ch（2バイト読み出しアクセス）を書込。

START	Slave Address	ACK	Access Address	ACK
S	D8h (6Ch (7b)+ 0)	A	00h	A

Reg Address H	ACK	Reg Address L	ACK	Serial Ctrl	ACK	STOP
D0h	A	51h	A	2Ch	A	P

I2C コマンド: リードバッファレジスタ（07h, 08h）で流量データ2バイトを読み出す。

START	Slave Address	ACK	Access Address	ACK
S	D8h (6Ch (7b)+ 0)	A	07h	A

Re-Start	Slave Address	ACK	Read Data H	ACK	Read Data L	ACK	STOP
RS	D9h (6Ch (7b)+ 1)	A	xxh	A	xxh	NA	P

読み出した 16bit のデータは符号なしデータです。これを Pv として、次の式で差圧(Pa)に変換します。

○ 差圧測定レンジ ±50[Pa] モデルの場合

*Dp[Pa] = (Pv - 1024)/60000*RANGE – RANGE/2* (RANGE: 100)

○ 差圧測定レンジ ±500[Pa] モデルの場合

*Dp[Pa] = (Pv - 1024)/60000*RANGE – RANGE/2* (RANGE: 1000)

○ 差圧測定レンジ 0-250[Pa] モデルの場合

*Dp[Pa] = (Pv - 1024)/60000*RANGE* (RANGE: 250)

4.CRC 演算機能有効化(任意)

CRC演算制御レジスタのビット1を"1"にセットします。CRC値読み出しシーケンスは7.4参照。

I2C コマンド: アクセスアドレスレジスタ(00h)を介してCRC演算制御レジスタ(D049h)に02h書込 (CRC_EN = 1)

START	Slave Address	ACK	Access Address	ACK	Reg Address H	ACK	Reg Address L	ACK
S	D8h (6Ch (7b)+ 0)	A	00h	A	D0h	A	49h	A

Serial Ctrl	ACK	Write Data	ACK	STOP
18h	A	02h	A	P

5.ハードウェアリセット実行

パワーシーケンスレジスタのビット7を"1"にセットします。 なお、ハードウェアリセット実行後は、パワーシーケンスレジスタのビット7は自動で"0"クリアされます。

I2C コマンド: パワーシーケンスレジスタ (0Dh) に 80h を書込 (Hard_Reset = 1)

START	Slave Address	ACK	Access Address	ACK	Write Data	ACK	STOP
S	D8h (6Ch (7b)+ 0)	A	0Dh	A	80h	A	P

7.3 レジスタ詳細説明

D6F-PH はコンフィギュレーションレジスタを介して通信が実行されます。

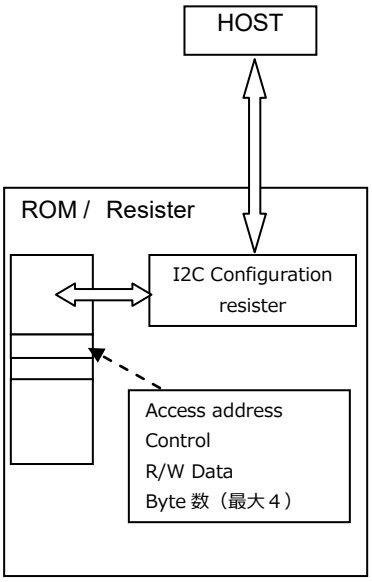


図 15 コンフィギュレーション概要

表 8 コンフィギュレーションレジスター覧

Configuration アドレス	レジスタ名	補足説明
00h	アクセスアドレス 1 (上位バイト)	最初のアクセスアドレスの上位バイト
01h	アクセスアドレス 2 (下位バイト)	最初のアクセスアドレスの下位バイト
02h	シリアルコントロール	ライト・リードアクセス制御
03h	ライトバッファ 0	アクセスアドレスに書かれるデータ
04h	ライトバッファ 1	アクセスアドレス+ 1 に書かれるデータ
05h	ライトバッファ 2	アクセスアドレス+ 2 に書かれるデータ
06h	ライトバッファ 3	アクセスアドレス+ 3 に書かれるデータ
07h	リードバッファ 0	アクセスアドレスからの読み出しデータ
08h	リードバッファ 1	アクセスアドレス+1 からの読み出しデータ
09h	リードバッファ 2	アクセスアドレス+2 からの読み出しデータ
0Ah	リードバッファ 3	アクセスアドレス+3 からの読み出しデータ
0Bh	初期化レジスタ	電源投入直後の初期化制御
0Dh	パワーシーケンス	ハードウェアリセット機能制御

上位バイト：16bit データの bit[15:8]の 1 バイト、下位バイト：16bit データの bit[7:0]の 1 バイト

7.3.1 アクセスアドレスレジスタ (00h – 01h)

アクセスアドレスレジスタは、内部レジスタにアクセスする為に使用します。このレジスタは複数バイト転送時にはアドレスが自動インクリメントされるので、スタートアドレスを指定します。

表 9 アクセスアドレスレジスタ

アドレス	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
00h	A15	A14	A13	A12	A11	A10	A9	A8
01h	A7	A6	A5	A4	A3	A2	A1	A0

内部レジスタは下記の種類があります。内部レジスタのアドレスは 16 bit で構成されており、このアドレスをアクセスアドレスレジスタ(00h と 01h)に書き込む事で、アクセスが可能になります。

表 10. 内部レジスター覧

アドレス	レジスタ名	レジスタ説明
D040h	SENS_CTRL	センサ制御レジスタ
D046h	FLAGS	フラグレジスタ
D049h	INT_CTRL	CRC 演算制御レジスタ
D051h	COMP_DATA1_H	補正流量値レジスタ
D052h	COMP_DATA1_L	
D061h	TMP_H	内部温度センサ値レジスタ
D062h	TMP_L	

表 11 SENS_CTRL (D040h)：センサ制御レジスタ

アドレス	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
D040h						MS	DV_PWR[1]	DV_PWR[0]
Write Access	None	None	None	None	None	Host & MCU	Host & MCU	Host & MCU
Default	0	0	0	0	0	0	0	0

DV_PWR[1：0] – メインデバイスパワーモード設定

0 0 = スタンバイ – 全てのブロックはパワーダウン状態です。

1 0 = MCU オン – MCU ブロックを動作時に必要です。 アナログ部、メモリ部への電源がオン状態となり、MCU へのクロック供給が開始されます。このレジスタは測定中に状態を変更しないで下さい。

MS – MCU スタート – DV_PWR の状態に応じて、MCU モードを実行します。

0 = ストップ - 測定処理シーケンスがストップし、各ブロックが電源オフ状態となります。

1 = スタート - MCU のクロック供給が開始され、設定に応じた MCU モードが実行されます。

表 12 FLAGS (D046h)：フラグレジスタ

アドレス	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
D046h					OS1		HV1	SV
Write Access	None	None	None	None	Host & MCU	None	Host & MCU	Host & MCU
Default				0	0	0	0	0

SV – 供給電圧(VDD) フラグ: 0 = 供給電圧は仕様範囲内。 1 = 供給電圧は仕様範囲外です。

HV1 – ヒーター供給電圧フラグ: 0 = ヒーター供給電圧は仕様範囲内。 1 = ヒーター供給電圧は仕様範囲外。

OS1 – センサ接続検知フラグ: 0 = センサは接続されています。 1 = センサは接続されていません。

＊その他のビットへのアクセス時は必ず"0"を設定して下さい。フラグレジスタ読出時は、MCU 更新との競合を回避する為 2 度読み出しを推奨します。

表 13 INT_CTRL (D049h): CRC 演算制御レジスタ

アドレス	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
D049h							CRC_EN	
Write Access	NONE	NONE	NONE	NONE	NONE	NONE	Host＆MCU	NONE
Default	0	0	0	0	0	0	1	0

CRC_EN – CRC チェック機能選択 （CRC は 2 バイト読出のみに対応。CRC 演算の詳細は 7.4 を参照して下さい。）

0 = CRC チェック演算機能を無効にする 1 = CRC チェック演算機能を有効にする

表 14 補正流量値レジスタ (D051h, D052h), 内部温度センサ値レジスタ(D061h, D062h)

アドレス	レジスタ名称	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	説明
D051h	COMP_DATA1_H	DATA<15:8>								補正後差圧データ
D052h	COMP_DATA1_L	DATA<7:0>								
D061h	TMP_H	DATA<15:8>								内部温度センサデータ
D062h	TMP_L	DATA<7:0>								

・COMP_DATA1_H と COMP_DATA1_L [D051h – D052h]：補正後差圧データ (符号無し:Uint16)

○ ±50[Pa]、±500[Pa]モデルの場合

*Dp[Pa] = (Pv - 1024)/60000*RANGE – RANGE/2* (RANGE: 100 or 1000, Pv: D051h – D052h の値)

○ 0-250[Pa]モデルの場合

*Dp[Pa] = (Pv - 1024)/60000*RANGE* (RANGE: 250, Pv：D051h – D052h の値)

・TMP_H と TMP_L [D061h – D062h]：ASIC内部温度センサデータ (符号つき:Int16)

Tv [℃] = (Rv – 10214) / 37.39 (Rv：D061h – D062h の値)

注記: 温度データは参考値としてのみご使用下さい。温度精度に関しては仕様上保証されていません。

7.3.2 シリアルコントロールレジスタ (02h)

表 15 シリアルコントロールレジスタ (02h)

アドレス	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
02h	D_byte_cnt[3]	D_byte_cnt[2]	D_byte_cnt[1]	D_byte_cnt[0]	Req	R_WZ	Acc_ctl2[1]	Acc_ctl2[0]

- Acc_ctl2 [1：0] – アクセスコントロールビット:
  0 0 = 16 ビットアドレスアクセス(A15-A0) (内部 ROM やレジスタなど)
  0 1 = 8 ビットアドレスアクセス(A7-A0) MCU 内部 256 バイト Dual ポート RAM アクセス
  1 0 = reserved
  1 1 = reserved
- R_WZ – リード/ ライト 選択ビット
  0 = ライトアクセス
  1 = リードアクセス
- Req- リクエストビット
  0 = 直前のアクセスリクエストは完了
  1 = 新規リクエスト 内部のシリアルバスブリッジ制御回路がリクエスト処理を完了すると、この Req フラグを"0"にクリアします。ライトアクセスリクエストの場合、内部バスブリッジ制御回路はライトバッファ内のデータをアクセスアドレスで指定されたレジスタに転送します。リードアクセスの場合、内部バスブリッジ制御回路は指定されたアドレスのデータをリードバッファに格納します。
- D_byte_cnt3 [3：0]
  転送バイト数指定 1, 2, 3, 4 バイト転送のみサポートします。

7.3.3 ライトバッファレジスタ (03h – 06h)

内部レジスタに値を書き込むための 4 つのライトバッファです。以下の 2 通りの方法で書き込む事ができます。下記は、data[0]を内部レジスタの Address=A[15:0]のレジスタに書込む例です。下記の 18h はシリアルコントロールレジスタ(02h)で 18h(1 byte の新規書き込み)をライトすることを意味します。

（方法 1）

下記の 5byte をバースト書き込みします。

・00h、A[15:8]、A[7:0]、18h、data[0]

＊最初の 00h はアクセスアドレスレジスタ(00h)です。

(方法 2)

2byte ずつ順番にライトします。

・00h、A[15:8]

・01h、A[7:0]

・03h、data [0]

・02h、18h

＊シリアルコントロールレジスタ(02h)をリードすると、bit[3]が"0"ならライトアクセス完了を意味します。

7.3.4　リードバッファレジスタ（07h – 0Ah)

内部レジスタの値を読み出すための４つのリードバッファです。以下の２通りの方法で読み出す事ができます。下記は、内部レジスタの Address=A[15:0]のレジスタの値を 1byte 読み出す例です。下記の 1Ch はシリアルコントロールレジスタ(02h)で 1Ch(1 byte の新規読み出し)をライトすることを意味します。

（方法 1）

下記の 4byte をバースト書き込みして、読み出しリクエストを発行します。

・00h、A[15:8]、A[7:0]、1Ch

リードリクエスト処理完了後、リードバッファ 07h を読み出します。

＊リードリクエスト処理が完了すると、シリアルコントロールレジスタ(02h)のリクエストビットが”0”クリアされます。

（方法 2）

2byte ずつ順番にライトします。

・00h、A[15:8]

・01h、A[7:0]

・02h、1Ch

リードリクエスト処理完了後、リードバッファ 07h を読み出します。

7.3.5 初期化レジスタ (0Bh)

電源投入後、初期化レジスタ(0Bh)に 00h をライト必要です。(NVM のトリミングデータをロードする為)

7.3.6 パワーシーケンスレジスタ (0Dh)

表 16 パワーシーケンスレジスタ (0Dh)

アドレス	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0Dh	Hard_Reset	ADC_state	ADC_state	ADC_state	Pwr_seq_state5	Pwr_seq_state5	Pwr_seq_state5	Pwr_seq_state5

表 17 パワーシーケンスレジスタ詳細

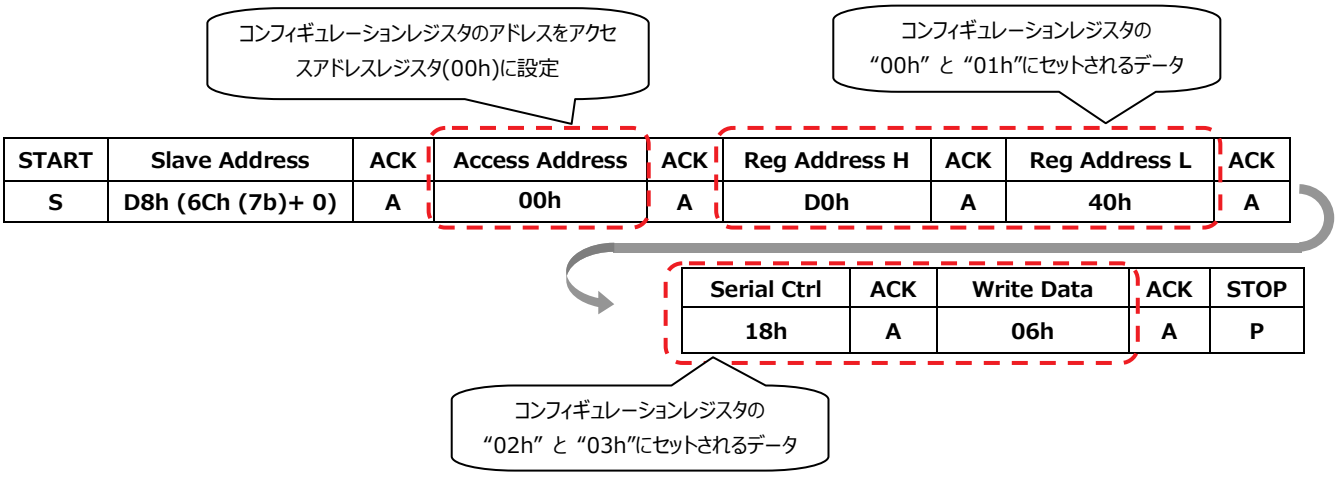
Bit	Name	R/W	Description
[3:0]	Pwr_seq_state5	R	パワーシーケンスのステート状態を示します。 h0(0000b): Idle (初期化処理後 or 電源リセット時) h2(0010b): Active (D040h に 06h を書込後) h9(1001b): Execute (演算中)
[6:4]	ADC_state	R	ADC を制御するステート状態
[7]	Hard_Reset	R/W	1->ハードウェアリセットを実行（実行後自動クリア） 0->ハードウェアリセットを実行しない。

ハードウェアリセットが実行された場合、ハードウェアリセットビットはリセット実行後に自動で”0”クリアされ、内部レジスタはデフォルト値に戻り、内部トリミング値は不揮発性メモリからリロードされます。このハードウェアリセット機能は、電源リセットと同様の機能となります。

なお、ハードウェアリセット使用時は、0～6 ビットは 0 としてください。

7.3.7 I2C アクセスコマンド例

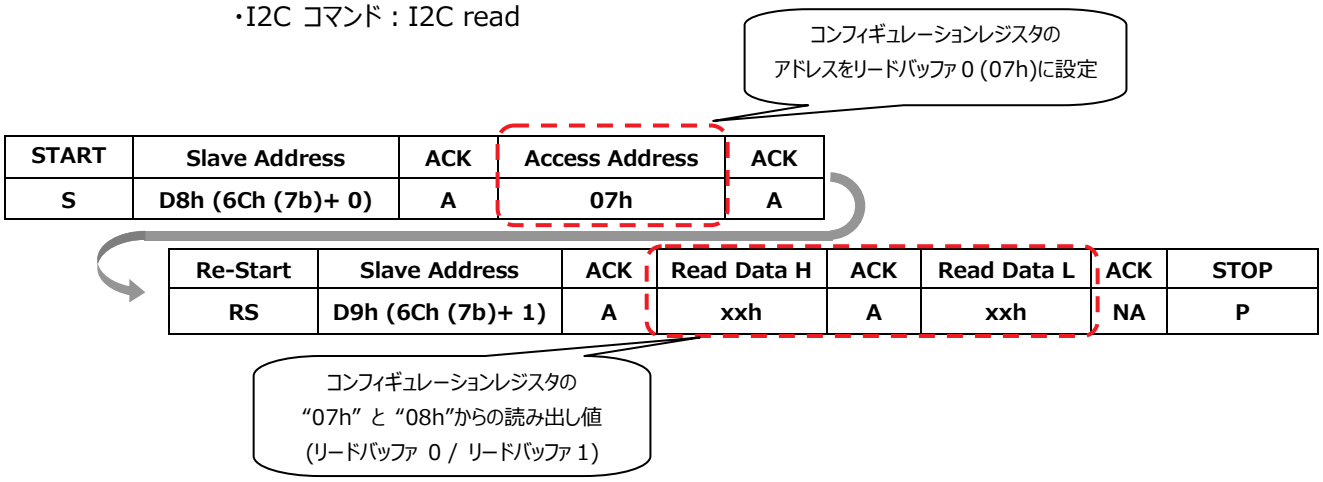
・I2C コマンド：I2C write



START	Slave Address	ACK	Access Address	ACK	Reg Address H	ACK	Reg Address L	ACK
S	D8h (6Ch (7b)+ 0)	A	00h	A	D0h	A	40h	A

Serial Ctrl	ACK	Write Data	ACK	STOP
18h	A	06h	A	P

・I2C コマンド：I2C read



START	Slave Address	ACK	Access Address	ACK
S	D8h (6Ch (7b)+ 0)	A	07h	A

Re-Start	Slave Address	ACK	Read Data H	ACK	Read Data L	ACK	STOP
RS	D9h (6Ch (7b)+ 1)	A	xxh	A	xxh	NA	P

7.4 CRC

・CRC 概要

CRC はデータ通信におけるエラー検出方法として用いられます。弊社フローセンサには CRC8 多項式 x^8 + x^5 + x^4 + 1 を用いています。CRC 機能を有効とした時の 2 バイト読み出しの I2C アクセス例を以下に示します。

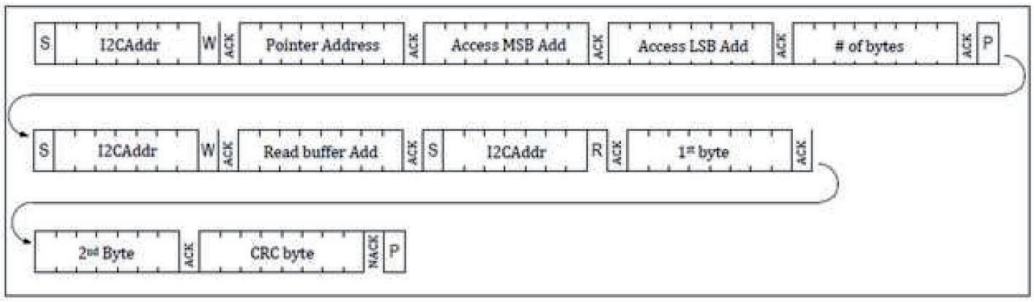


図 16. CRC 機能を有効とした時の 2 バイトデータ + CRC バイト読み出し例

・ビット単位 CRC-8 計算方法

1. データビット列を一列に並べます。
2. 多項式ビット列を、データビット列の下に並べます。
3. データビット列の一番左側のビットが"0"ならば、多項式ビット列を 1 ビット右にシフトします。データビット列の一番左側のビットが"1"ならば、多項式ビット列とで XOR 演算を実行します。その後多項式ビット列を 1 ビット右にシフトします。
4. 上記 1.～3.のステップを多項式ビット列がデータビット列の右端に到達するまで繰り返します。

以下に、XOR 演算に基づく CRC バイト算出例を示します。

	hex	bin
1st Byte of data	04h	00000100
2nd Byte of data	02h	00000010
Polynomial (x^8+ x^5+ x^4+ 1)	131h	100110001
CRC-byte checksum	225h	11100001

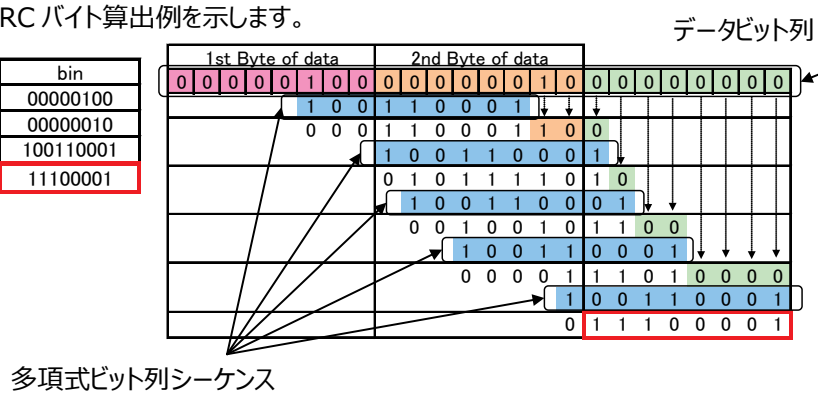


図 17. CRC-8 での XOR 演算例

8　開発ツール

ソフト開発支援ツールとして、3 種類のサンプルコードを用意しています。

1. Raspberry Pi 用　サンプルコード
2. Arduino 用　サンプルコード
3. STM32 MCU 用　サンプルコード

Raspberry Pi 用サンプルコードと Arduino 用サンプルコードは、オムロン評価ボードと一緒にご使用頂けます。オムロン評価ボードは下記の 3 種類のプラットフォームに対応しており、差圧センサ D6F-PH、評価ボード、ハーネスをプラットフォームに接続することで、簡単に評価することが可能です。
評価ボードは全ての D6F-PH に対応していますが、評価ボードと D6F-PH を接続するためのハーネスが D6F-PH の型式によって異なりますのでご注意ください。
評価ボード URL: (http://www.omron.co.jp/ecb/sensor/evaluation-board/2jcie)

表 18　評価ボード一覧

プラットフォーム	評価ボード	サンプルコード	接続用ハーネス（評価ボード・D6F-PH 間）	ハーネス対応差圧センサ形式
Raspberry Pi *1 接続用	形 2JCIE-EV01-RP1 	https://github.com/omron-devhub/d6f-2jcieev01-raspberrypi	形 2JCIE-HARNESS-02 *4	形 D6F-PH0025AD1
				形 D6F-PH0505AD3
				形 D6F-PH5050AD3
Arduino *2 接続用	形 2JCIE-EV01-AR1 	https://github.com/omron-devhub/d6f-2jcieev01-arduino	形 2JCIE-HARNESS-03 *5	形 D6F-PH0025AD2
				形 D6F-PH0505AD4
				形 D6F-PH5050AD4
ESP32 Feather *3 接続用	形 2JCIE-EV01-FT1 	https://github.com/omron-devhub/d6f-2jcieev01-arduino		形 D6F-PH0025AMD2
				形 D6F-PH0505AMD4
				形 D6F-PH5050AMD4

＊1. Raspberry Pi は、Raspberry Pi 財団の登録商標です。
＊2. Arduino は、Arduino LLC および Arduino SRL の登録商標です。
＊3. Feather は、Adafruit Industries LLC の登録商標です。
＊4. 形 2JCIE-HARNESS-02 は、片側はコネクタ、もう片側はリード線です。リード線を D6F-PH と接続して使用頂く必要があります。
＊5. 形 2JCIE-HARNESS-03 は、両側コネクタです。D6F-PH と評価ボードの両方をコネクタにて簡単に接続可能です。

評価ボードをご使用頂かなくても、サンプルコードは利用可能です。ただし、お客様でセンサを配線頂く必要があります。
なお、サンプルコードは評価用目的のためのコードであり、オムロンは動作保証しておりません。

8.1 Raspberry Pi 用サンプルコード

Raspberry Pi 用のサンプルコードは下記の URL にあります。

Github URL: https://github.com/omron-devhub/d6f-2jcieev01-raspberrypi

--差圧測定レンジ　0～250Pa 用

https://github.com/omron-devhub/d6f-2jcieev01-raspberrypi/blob/master/d6f-ph0025.c

--差圧測定レンジ　±50Pa 用

https://github.com/omron-devhub/d6f-2jcieev01-raspberrypi/blob/master/d6f-ph0505.c

--差圧測定レンジ　±500Pa 用

https://github.com/omron-devhub/d6f-2jcieev01-raspberrypi/blob/master/d6f-ph5050.c

サンプルコードの構成は下記の通りです。

```
/** <!-- main - Differential pressure sensor {{{1 -->
 * 1. read and convert sensor.
 * 2. output results, format is: [Pa]
 */
int main() {
    // 1. Initialize sensor (0Bh, 00h)
    i2c_write_reg16(D6F_ADDR, 0x0B00, NULL, 0);
    delay(900);
    while(1){
        // 2. Trigger getting data (00h, D0h, 40h, 18h, 06h)
        uint8_t send0[] = {0x40, 0x18, 0x06};
        i2c_write_reg16(D6F_ADDR, 0x00D0, send0, 3);

        delay(50);  // wait 50ms

        // 3. Read data (00h, D0h, 51h, 2Ch) (07h)
        uint8_t send1[] = {0x51, 0x2C};
        i2c_write_reg16(D6F_ADDR, 0x00D0, send1, 2);
        uint8_t rbuf[2];
        uint32_t ret = i2c_read_reg8(D6F_ADDR, 0x07, rbuf, 2);

        uint16_t rd_flow = conv8us_u16_be(rbuf);
        float flow_rate;

        // calculation for +/-50[Pa] range
        flow_rate = ((float)rd_flow - 1024.0) * 100.0 / 60000.0 - 50.0;

        printf("%6.2f", flow_rate);
        printf(" [Pa]\n");
    }
}
```

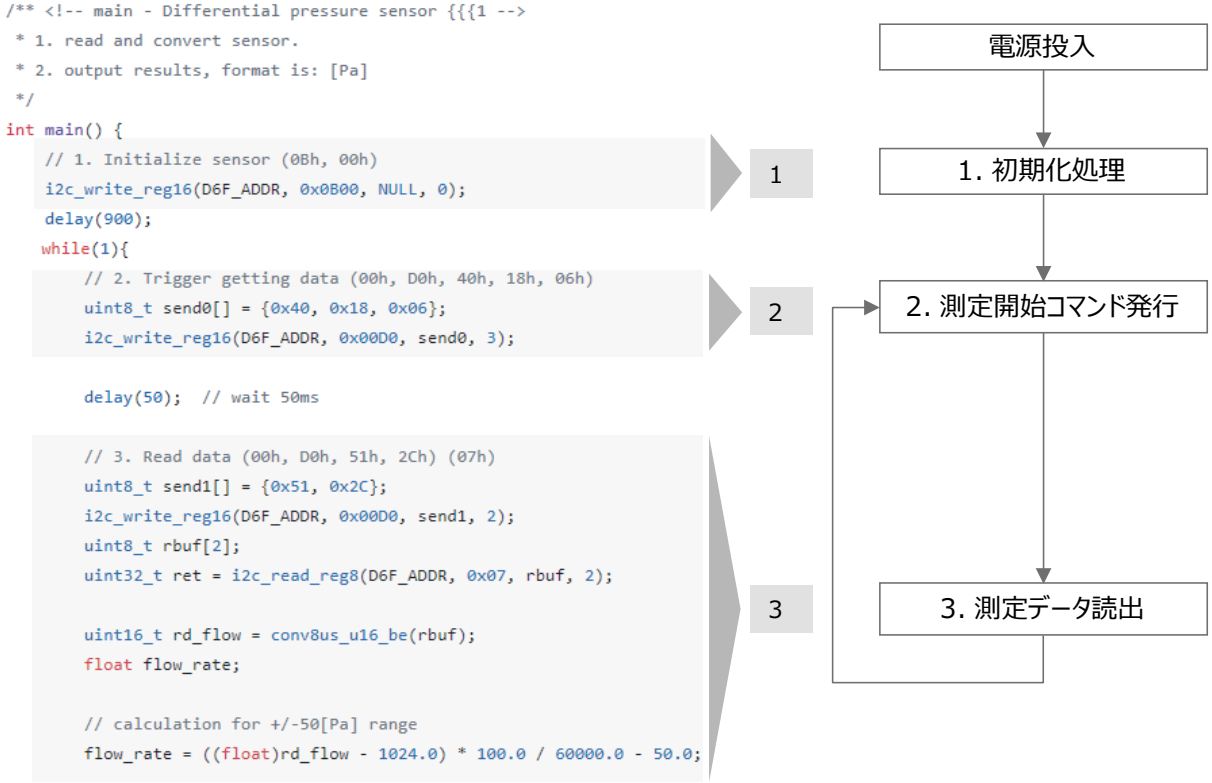


図 18　Raspberry Pi 用サンプルコード構成

Raspberry Pi 用サンプルコードの動作手順は下記の通りです。

(1) D6F-PH、ハーネス、オムロン評価ボード (2JCIE-EV01-RP1) を Raspberry Pi を接続

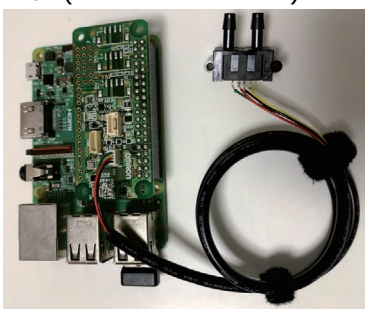


図 19 Set-up

(2) I2C を有効化

Raspberry Pi を立ち上げ、Start メニューから、"Preferences" > "Raspberry Pi Configuration" を開き、I2C を"Enable"し、再起動。

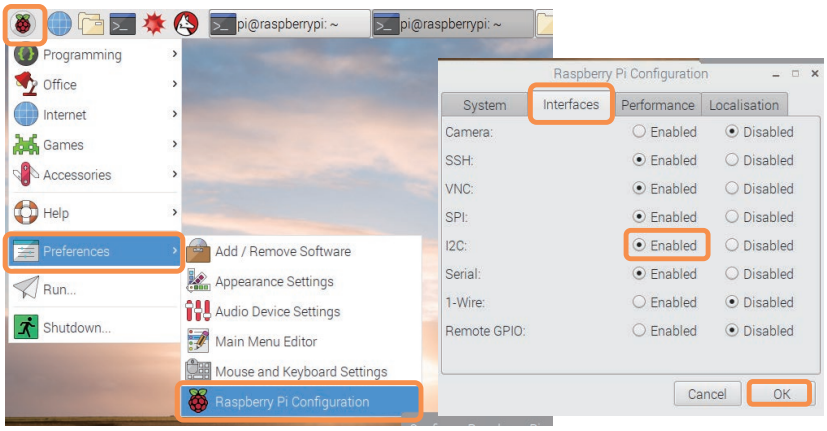


図 20 Enable I2C

(3) サンプルコードをダウンロード

下記の URL から Github にアクセスして、Zipfile をダウンロード。

GitHub URL: https://github.com/omron-devhub/d6f-2jcieev01-raspberrypi

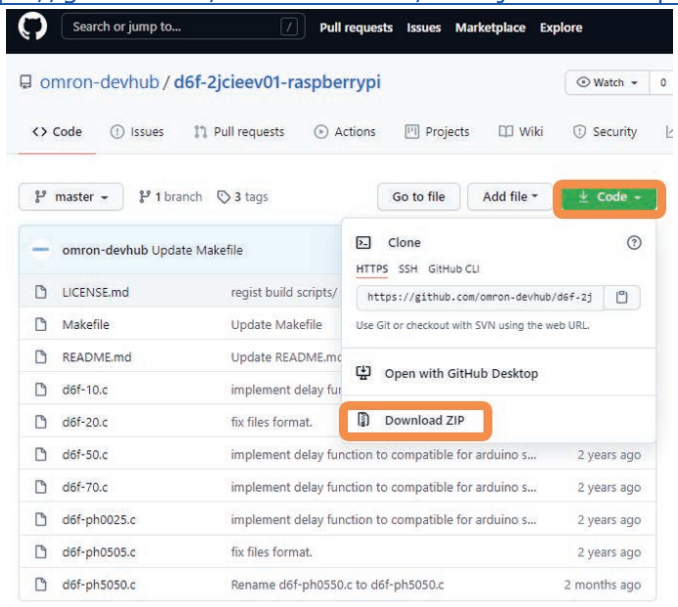


図 21 Download sample code

または、Terminal を開き、下記のコマンドを実行。

```
~$ git clone https://github.com/omron-devhub/d6f-2jcieev01-raspberrypi
```

または、インターネットに接続された他の PC で Zip file をダウンロードした後, USB メモリ等で Raspberry Pi に zip file を移動させる。

(4) Make file

Terminal を開き、下記のコマンドを実行。

```
pi@raspberrypi:~ $ cd Downloads/
pi@raspberrypi:~/Downloads $ unzip d6f-2jcieev01-raspberrypi-master.zip
pi@raspberrypi:~/Downloads $ cd d6f-2jcieev01-raspberrypi-master/
pi@raspberrypi:~/Downloads/d6f-2jcieev01-raspberrypi-master $ make all
```

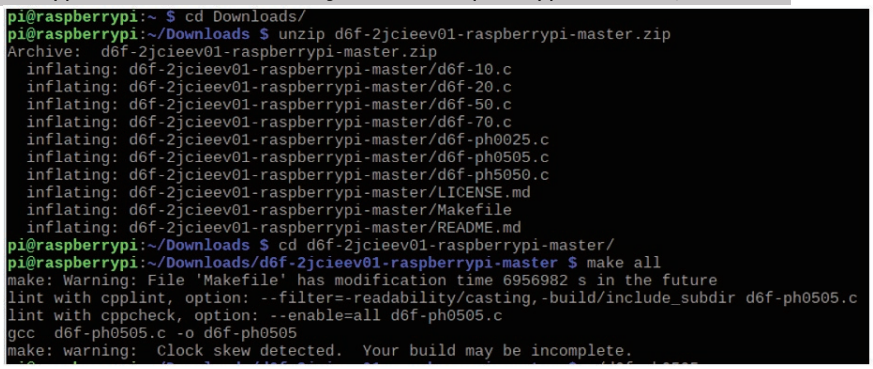


図 22 Make file

(5) Run the file

データ取得するために下記のコマンドを実行。

--差圧測定レンジ 0～250Pa の場合

```
pi@raspberrypi:~/Downloads/d6f-2jcieev01-raspberrypi-master $ ./d6f-ph0025
```

--差圧測定レンジ ±50Pa の場合

```
pi@raspberrypi:~/Downloads/d6f-2jcieev01-raspberrypi-master $ ./d6f-ph0505
```

--差圧測定レンジ ±500Pa の場合

```
pi@raspberrypi:~/Downloads/d6f-2jcieev01-raspberrypi-master $ ./d6f-ph5050
```

(データ取得を停止したい場合は、"Ctrl"キーと"C"キーを同時に押してください。)

図 23 Run the file

8.2 Arduino 用サンプルコード

Arduino 用のサンプルコードは下記の URL にあります。

Github URL: https://github.com/omron-devhub/d6f-2jcieev01-arduino

--差圧測定レンジ　0～250Pa 用

https://github.com/omron-devhub/d6f-2jcieev01-arduino/tree/master/examples/d6f-ph0025

--差圧測定レンジ　±50Pa 用

https://github.com/omron-devhub/d6f-2jcieev01-arduino/tree/master/examples/d6f-ph0505

--差圧測定レンジ　±500Pa 用

https://github.com/omron-devhub/d6f-2jcieev01-arduino/tree/master/examples/d6f-ph5050

サンプルコードの構成は下記の通りです。

```
void setup() {
    Serial.begin(115200);
    Serial.println("peripherals: I2C");
    Wire.begin();  // i2c master

    Serial.println("sensor: Differential pressure Sensor");
    delay(32);

    // 1. Initialize sensor (0Bh, 00h)
    i2c_write_reg16(D6F_ADDR, 0x0B00, NULL, 0);
}

/** <!-- loop - Differential pressure sensor {{{1 -->
 * 1. read and convert sensor.
 * 2. output results, format is: [Pa]
 */
void loop() {
    delay(900);

    // 2. Trigger getting data (00h, D0h, 40h, 18h, 06h)
    uint8_t send0[] = {0x40, 0x18, 0x06};
    i2c_write_reg16(D6F_ADDR, 0x00D0, send0, 3);

    delay(50);  // wait 50ms

    // 3. Read data (00h, D0h, 51h, 2Ch) (07h)
    uint8_t send1[] = {0x51, 0x2C};
    i2c_write_reg16(D6F_ADDR, 0x00D0, send1, 2);
    uint8_t rbuf[2];
    if (i2c_read_reg8(D6F_ADDR, 0x07, rbuf, 2)) {  // read from [07h]
        return;
    }
    uint16_t rd_flow = conv8us_u16_be(rbuf);
    float flow_rate;

    // calculation for +/-50[Pa] range
    flow_rate = ((float)rd_flow - 1024.0) * 100.0 / 60000.0 - 50.0;

    Serial.print(flow_rate, 2);  // print converted flow rate
    Serial.println(" [Pa]");
}
```

図 24　Arduino 用サンプルコード構成

Arduino 用サンプルコードの動作手順は下記の通りです。

(1) D6F-PH、ハーネス、オムロン評価ボード（2JCIE-EV01-AR1）を Arduino に接続

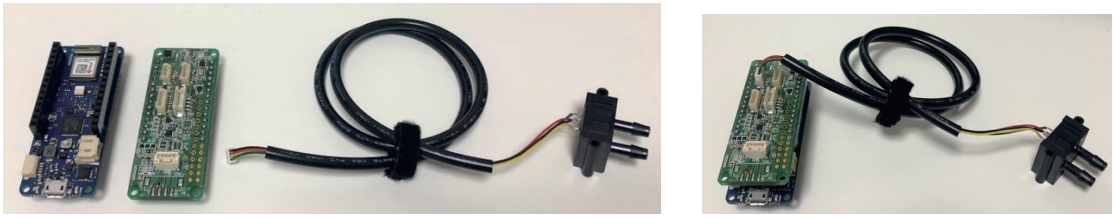


図 25 Set up

(2) Arduino IDE をダウンロード

下記 UARL から Arduino IDE をダウンロード。

https://www.arduino.cc/en/Main/Software

(3) Arduino IDE 上で Arduino を認識

Arduino IDE を開き、Arduino を PC に USB 経由で接続。

下記のような表示が出た場合、Arduino MKR 用の Package をインストール。

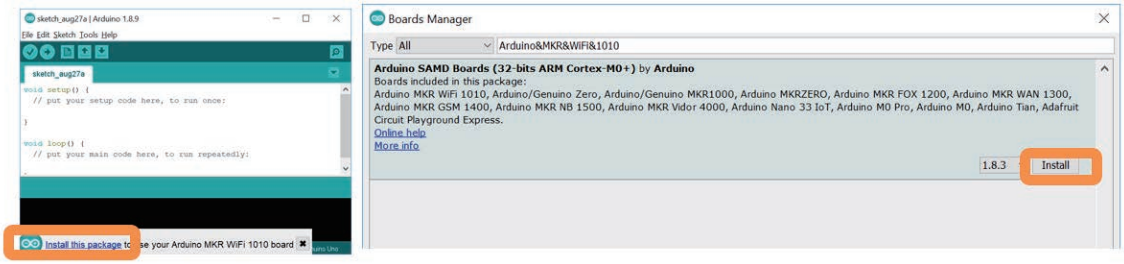


図 26 Arduino IDE

Driver をインストール。

図 27 Driver

Windows のスタートメニューで “Device Manager” と検索。

PC に認識された Arduino の COM ポート番号を確認。

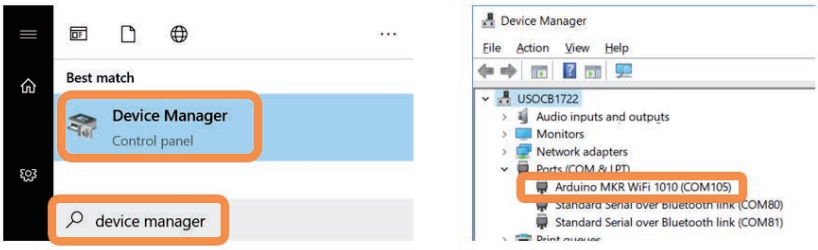


図 28 Device manager

“Tools” -> “Board Arduino” -> “Arduino MKR WiFi1010”.

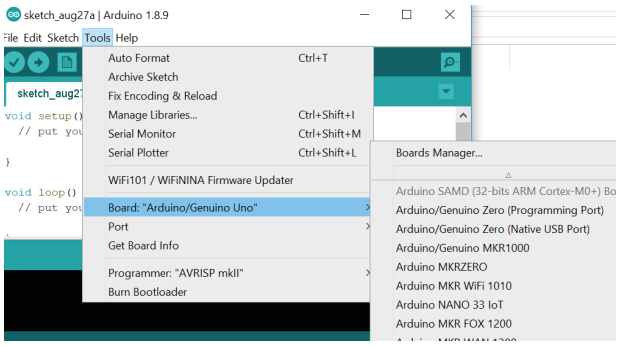


図 29 Select Arduino

“Tools” -> “PORT” -> Select COM port of Arduino.

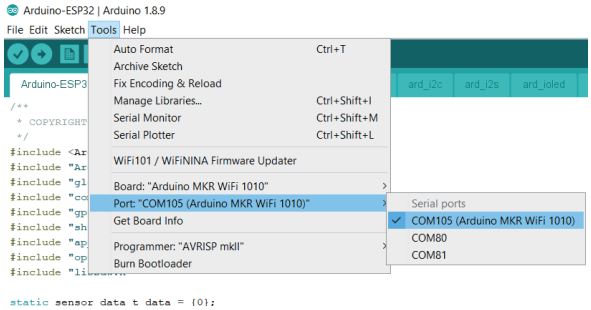


図 30 Select COM port

(4) サンプルコードをダウンロード

下記の GitHub の URL に接続して zip file をダウンロード。

GitHub URL: https://github.com/omron-devhub/d6f-2jcieev01-arduino

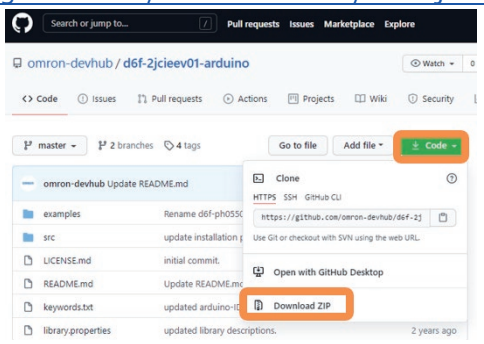


図 31 Download sample code

(5) サンプルコードを Arduino にアップロード。

“Sketch” -> “Include Library” -> “Add .ZIP Library”

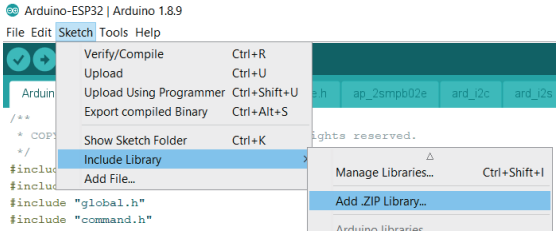


図 32 Add zip

“Downloads” フォルダを選択。“d6f-2jcieev01-arduino-master.zip”を選択。

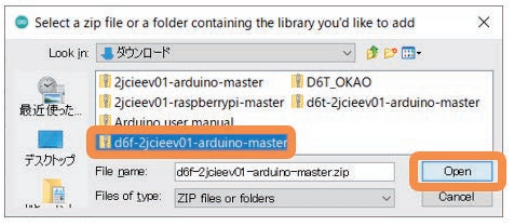


図 33 Select zip file

センサの差圧測定範囲に合わせて、ファイルを選択。

“File” -> “Examples” -> “D6F-2JCIE-EV01”

-> “d6f-ph0025” or “d6f-ph0505” or “d6f-ph5050”

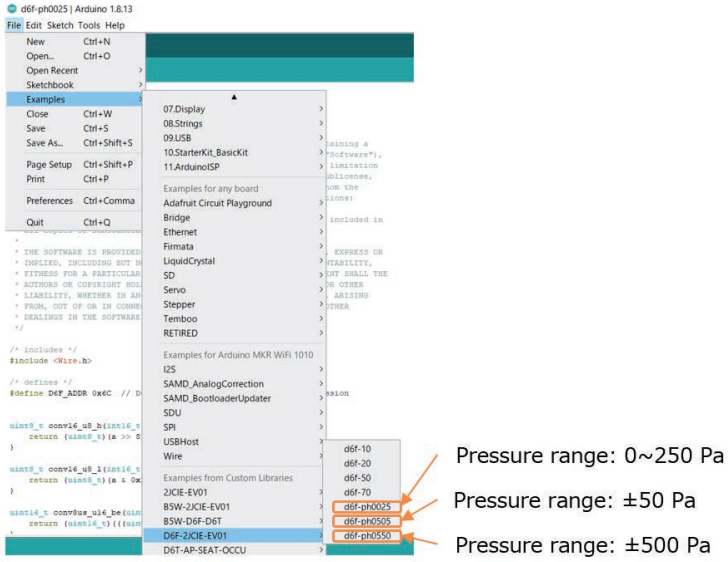


図 34 Select file

“Verify”をクリック。エラーが出ていないか確認。

図 35 Verify

“Upload”をクリック。“CPU reset” が表示されるか確認。

図 36 Upload

(6) データ取得

“Tools” -> “Serial Monitor”

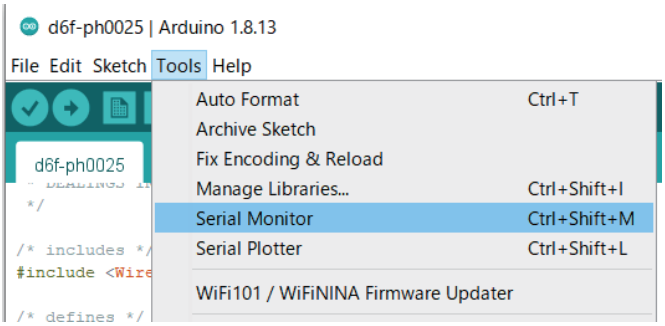


図 37 Serial monitor

データが表示される。

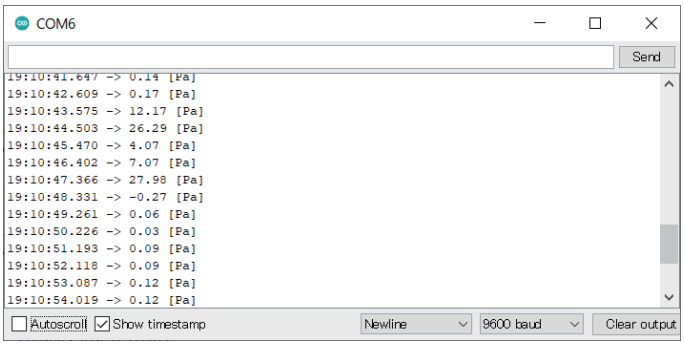


図 38 data

8.3 STM32 MCU 用サンプルコード

STM32 MCU 用のサンプルコードは一例です。実際にはお使いになる MCU の仕様に合わせて、I2C 制御の関数等を変更する必要があります。

使用する差圧測定レンジに合わせて、D6F_PH_Sample.h の下記の部分を変更してください。

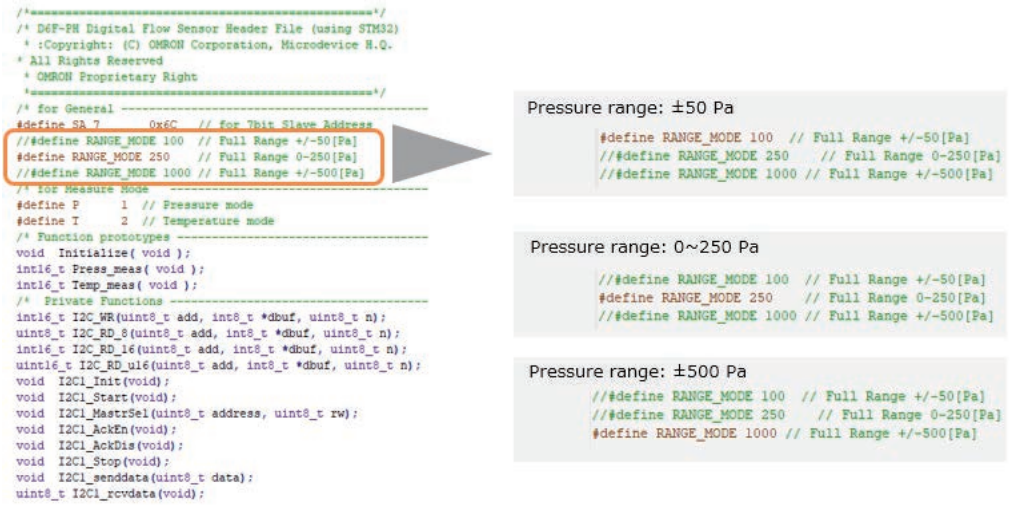


図 39 header file

D6F_PH_Sample.c のサンプルコードの構成は下記の通りです。

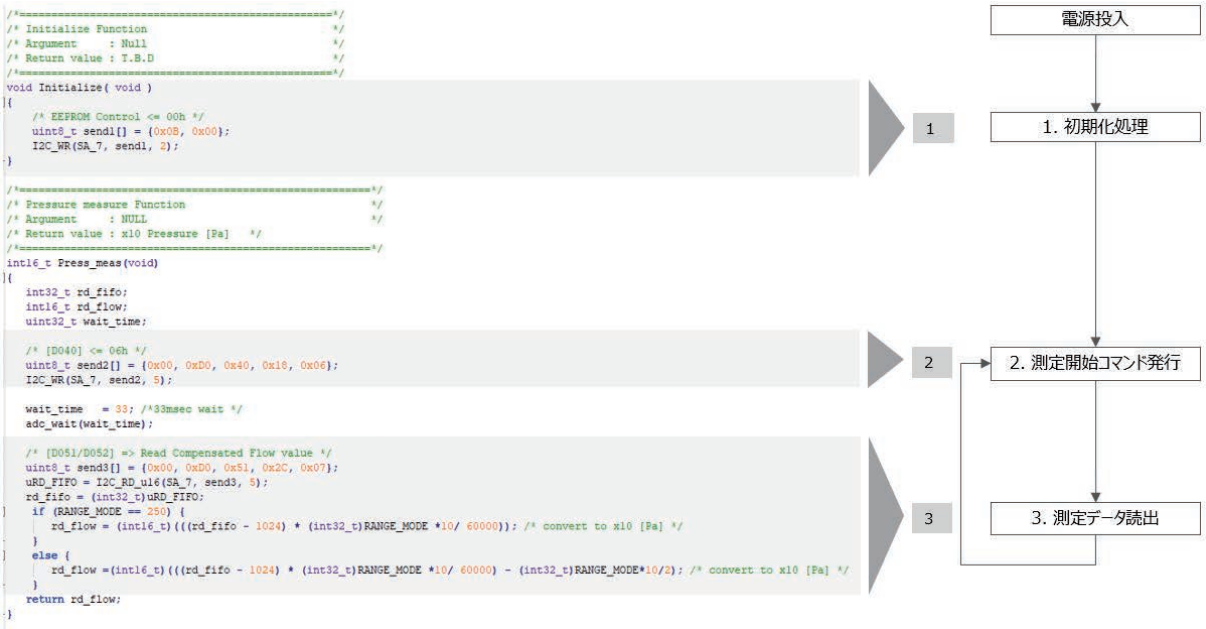


図 40 main code

このサンプルコードでは、センサから出力される 16bit 符号なし整数の値を、16 ビット符号つき整数（2 の補数）で 10 倍された圧力値[Pa]に変換しています。

例えば、16 ビット符号つき整数（2 の補数）で 10 倍された圧力値[Pa] rd_flow が 218 の時は、21.8Pa を意味します。

サンプルコード

D6F_PH_Sample.h

```
/*================================================*/
/* D6F-PH Digital Flow Sensor Header File (using STM32)
  * :Copyright: (C) OMRON Corporation, Microdevice H.Q.
  * :Auther     :
  * :Revision:   $Rev$
  * :Id:           $Id$
  * :Date:         $Date$
  *
  * All Rights Reserved
  * OMRON Proprietary Right
  *================================================*/
/*======================*/
/* for General                  */
/*======================*/
#define SA_7            0x6C     // for 7bit Slave Address
//#define RANGE_MODE 100    // Full Range +/-50[Pa]
#define RANGE_MODE 250       // Full Range 0-250[Pa]
//#define RANGE_MODE 1000 // Full Range +/-500[Pa]
/*======================*/
/* for Measure Mode          */
/*======================*/
#define P          1    // Pressure mode
#define T          2    // Temperature mode
/* Function prototypes -----------------------------------------------------*/
void    Initialize( void );
int16_t Press_meas( void );
int16_t Temp_meas( void );
/*    Private Functions -----------------------------------------------------*/
int16_t I2C_WR(uint8_t add, int8_t *dbuf, uint8_t n);
uint8_t I2C_RD_8(uint8_t add, int8_t *dbuf, uint8_t n);
int16_t I2C_RD_16(uint8_t add, int8_t *dbuf, uint8_t n);
uint16_t I2C_RD_u16(uint8_t add, int8_t *dbuf, uint8_t n);
void    I2C1_Init(void);
void    I2C1_Start(void);
void    I2C1_MastrSel(uint8_t address, uint8_t rw);
void    I2C1_AckEn(void);
void    I2C1_AckDis(void);
void    I2C1_Stop(void);
void    I2C1_senddata(uint8_t data);
uint8_t I2C1_rcvdata(void);
```

Please change the RANGE_MODE define for your target Product Pressure range.

D6F_PH_Sample.c

```
/*=================================================*/
/* D6F-PH Digital Flow Sensor Sample Code (using STM32)
  * :Copyright: (C) OMRON Corporation, Microdevice H.Q.
* All Rights Reserved
* OMRON Proprietary Right
  *=================================================*/
#include "stm32f10x_i2c.h"
#include "D6F_PH_Sample.h"

#define I2C1_SCL_PIN                    GPIO_Pin_6
#define I2C1_SDA_PIN                    GPIO_Pin_7
#define I2C2_SCL_PIN                    GPIO_Pin_10
#define I2C2_SDA_PIN                    GPIO_Pin_11

typedef unsigned char     uint8;
typedef unsigned short    uint16;
typedef unsigned long     uint32;
            int16_t RD_FIFO;    /* 16bit data width */
                        uint16_t uRD_FIFO; /* 16bit data width */
            uint8_t RD_REG;     /*    8bit data width */
            uint8_t setting_done_flag = 0;
//wait function
void adc_wait(volatile uint32_t delay)
{
                while(delay) delay--;
}
/*=================================================*/
/* Initialize Function                             */
/* Argument        : Null                          */
/* Return value : T.B.D                            */
/*=================================================*/
void Initialize( void )
{
                /* EEPROM Control <= 00h */
                uint8_t send1[] = {0x0B, 0x00};
                I2C_WR(SA_7, send1, 2);
}
/*=========================================================*/
/* Pressure measure Function                               */
/* Argument        : NULL                                  */
/* Return value : x10 Pressure [Pa]     */
/*=========================================================*/
int16_t Press_meas(void)
{
    int32_t rd_fifo;
    int16_t rd_flow;
    uint32_t wait_time;

    /* [D040] <= 06h */
    uint8_t send2[] = {0x00, 0xD0, 0x40, 0x18, 0x06};
    I2C_WR(SA_7, send2, 5);

    wait_time     = 33; /*33msec wait */
    adc_wait(wait_time);

    /* [D051/D052] => Read Compensated Flow value */
    uint8_t send3[] = {0x00, 0xD0, 0x51, 0x2C, 0x07};
    uRD_FIFO = I2C_RD_u16(SA_7, send3, 5);
    rd_fifo = (int32_t)uRD_FIFO;
                if (RANGE_MODE == 250) {
                    rd_flow = (int16_t)(((rd_fifo - 1024) * (int32_t)RANGE_MODE *10/ 60000)); /* convert to x10 [Pa] */
                }
                else {
                    rd_flow  =(int16_t)(((rd_fifo  -  1024)  *  (int32_t)RANGE_MODE  *10/  60000)  -  (int32_t)RANGE_MODE*10/2);  /*
convert to x10 [Pa] */
                }
    return rd_flow;
}
```

```
/*=======================================================*/
/* Temperature measure Function                          */
/* Argument        : NULL                                */
/* Return value : x10 Temperature [degC]                 */
/*=======================================================*/
int16_t Temp_meas(void)
{
    int16_t rd_temp;
    uint_32t wait_time;

    /* [D040] <= 06h */
    uint8_t send2[] = {0x00, 0xD0, 0x40, 0x18, 0x06};
    I2C_WR(SA_7, send2, 5);
    wait_time   = 33; /* 33msec wait */
    adc_wait(wait_time);

    /* [D061/D062] => Read TMP_H/TMP_L value */
    uint8_t send3[] = {0x00, 0xD0, 0x61, 0x2C, 0x07};
    RD_FIFO = I2C_RD_16 (SA_7, send3, 5);
    rd_temp = (int16_t)((((int32_t)RD_FIFO -10214)*1000 / 3739)); // convert to x10 [degC]
    return rd_temp;
}

/* Public Basic Functions ------------------------------------------------------*/
/*=================================================*/
/* I2C Write command                               */
/* Argument        : 7bit Slave Address(uint8_t)   */
/*                   int8_t *dbuf (Write data)     */
/*                   uint8_t n (Number of bytes)*/
/* Return value : 8bit Read result                 */
/*=================================================*/
int16_t I2C_WR(uint8_t add, int8_t *dbuf, uint8_t n) {
    int16_t i = 0;
    I2C1_Start();                   /* Start condition    */
    I2C1_MastrSel(add, 0);      /* Slave Address       */
    while (n--) {
      I2C1_senddata(dbuf[i]); /* Send    Data          */
      i++;
    }
    I2C1_Stop();                    /* Stop condition    */
    return 0;
}
/*=================================================*/
/* I2C uint_8 Read command                                 */
/* Argument        : uint8_t add (7bit Slave Address) */
/*                   int8_t *dbuf (Write data)         */
/*                   uint8_t n (Number of bytes)*/
/* Return value : 8bit Read result                 */
/*=================================================*/
uint8_t I2C_RD_8 (uint8_t add, int8_t *dbuf, uint8_t n) {
    int8_t i= 0;
    uint8_t n_w;
    n_w = n - 1;
    /* I2C Pre-WR Access */
    I2C1_Start();                   /* Start condition    */
    I2C1_MastrSel(add, 0);      /* Slave Address 7bit => 8bit */
    while (n_w--) {
      I2C1_senddata(dbuf[i]); /* Send    Data          */
      i++;
    }
    I2C1_Stop();                    /* Stop    condition    */
    /* I2C RD Access */
    I2C1_Start();                   /* Start condition    */
    I2C1_MastrSel(add, 0);      /* Slave Address 7bit => 8bit */
    I2C1_senddata(dbuf[n-1]); /* Word    Address       */
    I2C1_Start();                   /* Re-Start condition */
    I2C1_MastrSel(add, 1);      /* Slave 7bit => 8bit for RD */
    I2C1_AckDis();                  /* ack diable for 1 byte */
    I2C1_Stop();                    /* Stop condition send */
    RD_REG = I2C1_rcvdata();    /* Read    Data */
    return RD_REG;
}
```

```
/*================================================*/
/* I2C int_16 Read command                                    */
/* Argument        : int8_t add (7bit Slave Address)      */
/*                   int8_t *dbuf (Write data)               */
/*                   uint8_t n (Number of bytes)*/
/* Return value : 16bit Read result                       */
/*================================================*/
int16_t I2C_RD_16 (uint8_t add, int8_t *dbuf, uint8_t n) {
    int16_t i= 0;
    uint8_t n_w;
    uint8_t rd_fifo[2] = {0, 0};
    n_w = n - 1;
    /* I2C Pre-WR Access */
    I2C1_Start();                       /* Start condition    */
    I2C1_MastrSel(add, 0);              /* Slave Address 7bit => 8bit */
    while (n_w--) {
      I2C1_senddata(dbuf[i]);       /* Send    Data           */
      i++;
    }
    I2C1_Stop();                        /* Stop    condition    */

    adc_wait(5); /* 5msec wait */
    I2C1_Start();                       /* Start condition    */
    I2C1_MastrSel(add, 0);              /* Slave Address 7bit => 8bit */
    I2C1_senddata(dbuf[n-1]);       /* Word    Address        */
    I2C1_Start();                       /* Re-Start condition */
    I2C1_MastrSel(add, 1);              /* Slave 7bit => 8bit for RD */
    I2C1_AckEn();                       /* ack enable send after MSB 1 byte read */
    rd_fifo[0] = I2C1_rcvdata(); /* Read    Data */
    I2C1_AckDis();                      /* ack diable send after LSB 1 byte read */
    I2C1_Stop();                        /* Stop condition send */
    rd_fifo[1] = I2C1_rcvdata(); /* Read    Data */
    RD_FIFO = (int16_t)((((uint16_t)rd_fifo[0] << 8) | (uint16_t)rd_fifo[1]));
    return RD_FIFO;
}

/*================================================*/
/* I2C uint_16 Read command                                   */
/* Argument        : int8_t add (7bit Slave Address)      */
/*                   int8_t *dbuf (Write data)               */
/*                   uint8_t n (Number of bytes)*/
/* Return value : 16bit Read result                       */
/*================================================*/
uint16_t I2C_RD_u16 (uint8_t add, int8_t *dbuf, uint8_t n) {
    int16_t i= 0;
    uint8_t n_w;
    uint8_t rd_fifo[2] = {0, 0};
    n_w = n - 1;
    /* I2C Pre-WR Access */
    I2C1_Start();                       /* Start condition    */
    I2C1_MastrSel(add, 0);              /* Slave Address 7bit => 8bit */
    while (n_w--) {
      I2C1_senddata(dbuf[i]);       /* Send    Data           */
      i++;
    }
    I2C1_Stop();                        /* Stop    condition    */
    adc_wait(5); /* 5msec wait */

    I2C1_Start();                       /* Start condition    */
    I2C1_MastrSel(add, 0);              /* Slave Address 7bit => 8bit */
    I2C1_senddata(dbuf[n-1]);       /* Word    Address        */
    I2C1_Start();                       /* Re-Start condition */
    I2C1_MastrSel(add, 1);              /* Slave 7bit => 8bit for RD */
    I2C1_AckEn();                       /* ack enable send after MSB 1 byte read */
    rd_fifo[0] = I2C1_rcvdata(); /* Read    Data */
    I2C1_AckDis();                      /* ack diable send after LSB 1 byte read */
    I2C1_Stop();                        /* Stop condition send */
    rd_fifo[1] = I2C1_rcvdata(); /* Read    Data */
    uRD_FIFO = (((uint16_t)rd_fifo[0] << 8) | (uint16_t)rd_fifo[1]);
    return uRD_FIFO;
}
```

```
void I2C1_Init(){
		I2C_InitTypeDef	I2C1_InitStructure;

		RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C1, ENABLE);		// start clock of I2C
		I2C1_InitStructure.I2C_Mode = I2C_Mode_I2C;
		I2C1_InitStructure.I2C_DutyCycle = I2C_DutyCycle_2;
		I2C1_InitStructure.I2C_Ack = I2C_Ack_Enable;
		I2C1_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
		I2C1_InitStructure.I2C_ClockSpeed = 400000;

		GPIO_InitTypeDef GPIO_InitStructure;			// make instance of InitStructure
		RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);	// start clock of GPIO pins
		GPIO_InitStructure.GPIO_Pin =( I2C1_SCL_PIN | I2C1_SDA_PIN );
		GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
		GPIO_InitStructure.GPIO_Mode	= GPIO_Mode_AF_OD;
		GPIO_Init(GPIOB, &GPIO_InitStructure);

		I2C_DeInit(I2C1);
		I2C_Init(I2C1, &I2C1_InitStructure);	// Initialize	with above parameters
		I2C_Cmd(I2C1, ENABLE);
}

void I2C1_Start(){
		I2C_GenerateSTART(I2C1,ENABLE);	// issue start condition
		while(!I2C_CheckEvent(I2C1,I2C_EVENT_MASTER_MODE_SELECT));
}

void I2C1_MastrSel( uint8_t address, uint8_t RW){
    uint8_t	direct;
    uint32_t event;
    direct =(RW == 0)?I2C_Direction_Transmitter : I2C_Direction_Receiver;
    event		=(RW		==		0)?I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED		:
I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED;
    I2C_Send7bitAddress(I2C1,(address << 1),direct ); //write to Slave
    while(!I2C_CheckEvent(I2C1, event)); // wait ACK
}

void I2C1_senddata(uint8_t data){
  I2C_SendData(I2C1, data);					//Send receive command
  while(!I2C_CheckEvent(I2C1,I2C_EVENT_MASTER_BYTE_TRANSMITTED)); // wait ACK
}

uint8_t I2C1_rcvdata(void){
  while(!I2C_CheckEvent(I2C1,I2C_EVENT_MASTER_BYTE_RECEIVED)); // wait ACK
  return I2C_ReceiveData(I2C1);	// receive 4th 8bit data
}

void I2C1_Stop(){
  I2C_GenerateSTOP(I2C1, ENABLE);		// put stop condition
}

void I2C1_AckEn(){
  I2C_AcknowledgeConfig(I2C1, ENABLE);	// ack enable
}

void I2C1_AckDis(){
  I2C_AcknowledgeConfig(I2C1, DISABLE);	// ack disable
}
```

9　ご承諾事項

平素はオムロン株式会社(以下「当社」)の商品をご愛用いただき誠にありがとうございます。
「当社商品」のご購入については、お客様のご購入先にかかわらず、本ご承諾事項記載の条件を適用いたします。ご承諾のうえご注文ください。

1．定義

本ご承諾事項中の用語の定義は次のとおりです。

(1)「当社商品」：「当社」のＦＡシステム機器、汎用制御機器、センシング機器、電子・機構部品

(2)「カタログ等」：「当社商品」に関する、オムロン総合カタログ、ＦＡシステム機器総合カタログ、セーフティコンポ総合カタログ、電子・機構部品総合カタログその他のカタログ、仕様書、取扱説明書、マニュアル等であって電磁的方法で提供されるものも含みます。

(3)「利用条件等」：「カタログ等」に記載の、「当社商品」の利用条件、定格、性能、動作環境、取扱い方法、利用上の注意、禁止事項その他

(4)「お客様用途」：「当社商品」のお客様におけるご利用方法であって、お客様が製造する部品、電子基板、機器、設備またはシステム等への「当社商品」の組み込み又は利用を含みます。

(5)「適合性等」：「お客様用途」での「当社商品」の(a)適合性、(b)動作、(c)第三者の知的財産の非侵害、(d)法令の遵守および(e)各種規格の遵守

2．記載事項のご注意

「カタログ等」の記載内容については次の点をご理解ください。

(1) 定格値および性能値は単独試験における各条件のもとで得られた値であり、各定格値および性能値の複合条件のもとで得られる値を保証するものではありません。

(2) 参考データはご参考として提供するもので、その範囲で常に正常に動作することを保証するものではありません。

(3) 利用事例はご参考ですので、「当社」は「適合性等」について保証いたしかねます。

(4)「当社」は、改善や当社都合等により、「当社商品」の生産を中止し、または「当社商品」の仕様を変更することがあります。

3．ご利用にあたってのご注意

ご採用およびご利用に際しては次の点をご理解ください。

(1) 定格・性能ほか「利用条件等」を遵守しご利用ください。

(2) お客様ご自身にて「適合性等」をご確認いただき「当社商品」のご利用の可否をご判断ください。「当社」は「適合性等」は一切保証いたしかねます。

(3)「当社商品」がお客様のシステム全体の中で意図した用途に対して、適切に配電・設置されていることをお客様ご自身で必ず事前に確認してください。

(4)「当社商品」をご使用の際には、(i)定格および性能に対し余裕のある「当社商品」のご利用 (ii) 冗長設計など「当社商品」が故障しても「お客様用途」の危険を最小にする安全設計、(ⅲ)利用者に危険を知らせる安全対策をシステム全体として構築、(ⅳ)「当社商品」および「お客様用途」の定期的な保守の各事項を実施してください。

(5)「当社商品」は、一般工業製品向けの汎用品として設計製造されています。従いまして、次に掲げる用途での使用は意図しておらず、お客様が「当社商品」をこれらの用途に使用される際には、「当社」は「当社商品」に対して一切保証をいたしません。なお、昇降設備、医用機器など下記に例示されている用途であっても、その具体的なご利用方法によっては、一般工業製品向けの汎用品として次項に定める通常の保証が可能な場合がありますので、当社営業担当者にご相談ください。

(a) 高い安全性が必要とされる用途（例：原子力制御設備、燃焼設備、航空・宇宙設備、鉄道設備、昇降設備、遊園地機械、医用機器、安全装置、その他生命・身体に危険が及びうる用途）
(b) 高い信頼性が必要な用途（例：ガス・水道・電気等の供給システム、24 時間連続運転システム、決済システムほか権利・財産を取扱う用途など）
(c) 厳しい条件または環境での用途（例：屋外に設置する設備、化学的汚染を被る設備、電磁的妨害を被る設備、振動・衝撃を受ける設備など）
(d) 「カタログ等」に記載のない条件や環境での用途
(6) 上記 3.(5)(a)から(d)に記載されている他、「本カタログ等記載の商品」は自動車（二輪車含む。以下同じ）向けではありません。自動車に搭載する用途には利用しないで下さい。自動車搭載用商品については当社営業担当者にご相談ください。

4. 保証条件

「当社商品」の保証条件は次のとおりです。
(1) 保証期間　当社又は当社の代理店よりご購入後 1 年間といたします。
（ただし「カタログ等」に別途記載がある場合を除きます。）
(2) 保証内容　故障した「当社商品」について、以下のいずれかを「当社」の任意の判断で実施します。
(a) 当社保守サービス拠点における故障した「当社商品」の無償修理
（ただし、電子・機構部品については、修理対応は行いません。）
(b) 故障した「当社商品」と同数の代替品の無償提供
(3) 保証対象外　故障の原因が次のいずれかに該当する場合は、保証いたしません。
(a) 「当社商品」本来の使い方以外のご利用
(b) 「利用条件等」から外れたご利用
(c) 「当社」以外による改造、修理による場合
(d) 「当社」以外の者によるソフトウェアプログラムによる場合
(e) 「当社」からの出荷時の科学・技術の水準では予見できなかった原因
(f) 上記のほか「当社」または「当社商品」以外の原因（天災等の不可抗力を含む）

5. 責任の制限

本ご承諾事項に記載の保証が「当社商品」に関する保証のすべてです。「当社商品」に関連して生じた損害について、「当社」および「当社商品」の販売店は責任を負いません。

6. 輸出管理

「当社商品」または技術資料を輸出または非居住者に提供する場合は、安全保障貿易管理に関する日本および関係各国の法令・規制を遵守ください。お客様が、法令・規則に違反する場合には、「当社商品」または技術資料をご提供できない場合があります。以上

(EC300)

●本誌に記載のない条件や環境での使用、および原子力制御・鉄道・航空・車両・燃焼装置・医療機器・娯楽機械・安全機器、その他人命や財産に大きな影響が予測されるなど、特に安全性が要求される用途に使用される際には、当社の意図した特別な商品用途の場合や特別の合意がある場合を除き、当社は当社商品に対して一切保証をいたしません。
●本製品の内、外国為替及び外国貿易法に定める輸出許可、承認対象貨物（又は技術）に該当するものを輸出（又は非居住者に提供）する場合は同法に基づく輸出許可、承認（又は役務取引許可）が必要です。

オムロン株式会社 インダストリアルオートメーションビジネスカンパニー

●製品に関するお問い合わせ先
お客様相談室
フリー通話 0120-919-066（クイック／オムロン）
携帯電話・PHS・IP電話などではご利用いただけませんので、下記の電話番号へおかけください。
電話 055-982-5015（通話料がかかります）
■営業時間：8:00～21:00　■営業日：365日
●FAXやWebページでもお問い合わせいただけます。
FAX 055-982-5051 / www.fa.omron.co.jp

●その他のお問い合わせ
納期・価格・サンプル・仕様書は貴社のお取引先、または貴社担当オムロン販売員にご相談ください。
オムロン制御機器販売店やオムロン販売拠点は、Webページでご案内しています。

オムロン制御機器の最新情報をご覧いただけます。
www.fa.omron.co.jp
緊急時のご購入にもご利用ください。

オムロン商品のご用命は

Man. No. CDSC-025B　2021年3月現在