OMRON

MEMS差压传感器

D6F-PH

用户手册

MEMS差压传感器

CDSC-CN1-025B

目录

1　概要

本用户手册介绍了本公司 MEMS 差压传感器（D6F-PH）的使用方法和特别记载事项等。此外，本手册是产品样本的补充资料，在实际使用时，请参照产品样本一起使用。

2　构造

图 1 所示为 MEMS 差压传感器（D6F-PH）的内部剖面示意图。从压力导入孔进入的空气流通过流量传感器芯片表面，从相反侧的导入孔排出。此时根据 MEMS 流量传感器的工作原理检测气流。

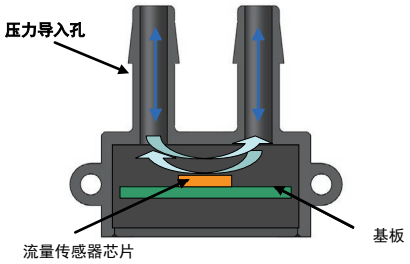


图 1　MEMS 差压传感器（D6F-PH）的内部剖面示意图

3　外形尺寸图

基板封装型
D6F-PH0025AD1
D6F-PH0505AD3
D6F-PH5050AD3

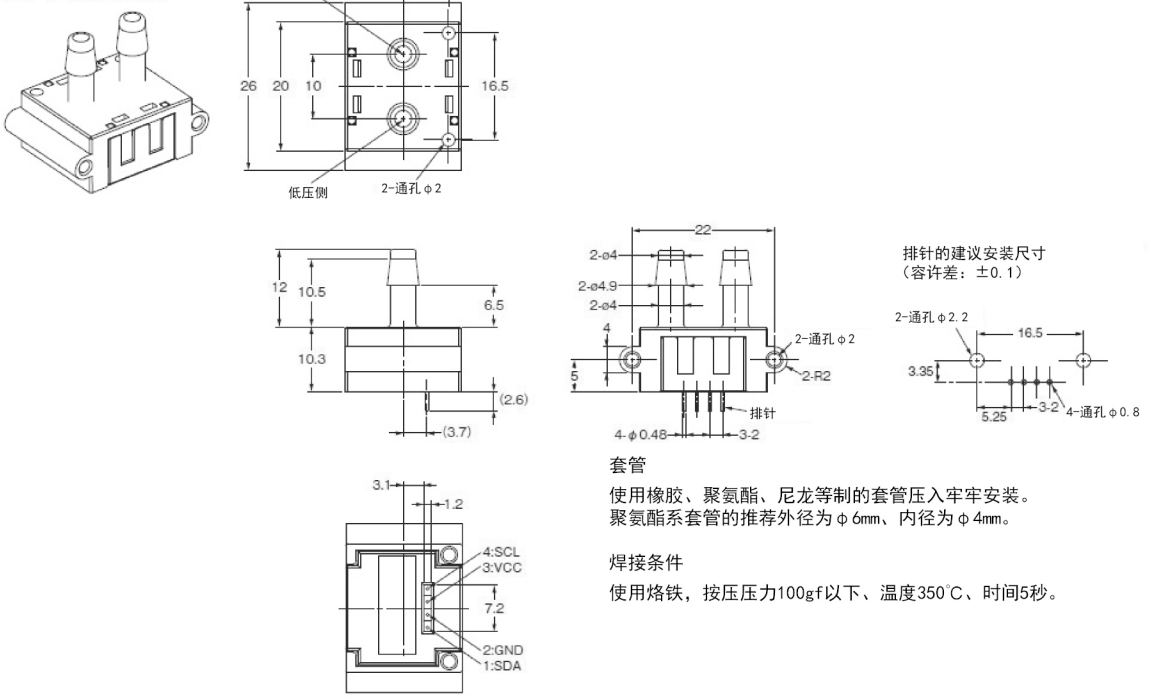
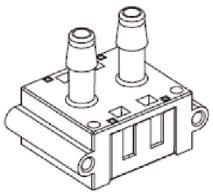
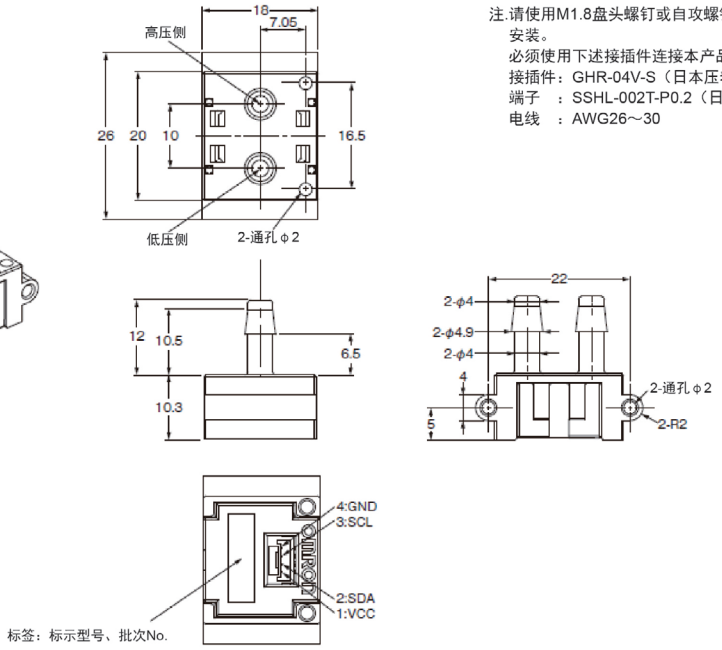


图 2　D6F-PH0505AD3 / D6F-PH0025AD1 / D6F-PH5050AD3 的外形尺寸图

接插件类型
D6F-PH0025AD2
D6F-PH0505AD4
D6F-PH5050AD4





注.请使用M1.8盘头螺钉或自攻螺钉，以0.36N·m以下的紧固扭矩进行安装。
必须使用下述接插件连接本产品。
接插件：GHR-04V-S（日本压着端子制造（株）生产）
端子　：SSHL-002T-P0.2（日本压着端子制造（株）生产）
电线　：AWG26～30

图 3 D6F-PH0505AD4 / D6F-PH0025AD2 / D6F-PH5050AD4 的外形尺寸图

D6F-PH0025AMD2
D6F-PH0505AMD4
D6F-PH5050AMD4

注1.安装螺钉使用M3圆头小螺钉或自攻螺钉，紧固扭矩控制在1.0N·m以下。
注2.螺钉头部及垫圈外径为6mm以下。
注3.导入端口的密封部位由φ7.0槽、P4 O型圈组成。（基于JISB 2401）
注4.必须使用下述接插件连接本产品。
接插件：GHR-04V-S（日本压着端子制造（株）生产）
端子　：SSHL-002T-P0.2（日本压着端子制造（株）生产）
电线　：AWG26～30

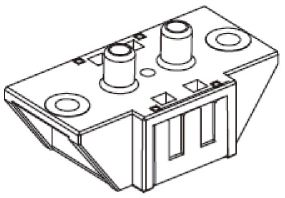


图 4 D6F-PH0505AMD4 / 6F-PH0025AMD2 / D6F-PH5050AMD4 的外形尺寸图

4　工作原理

MEMS 差压传感器（D6F-PH）使用热式质量流量传感器测量由设置在主流路的节流孔分流的气流，检测细微差压。

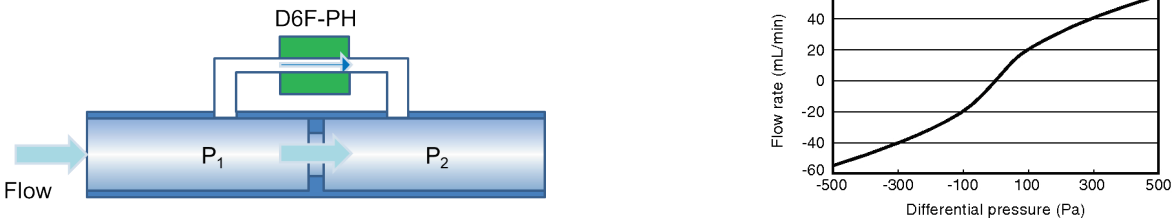


图 5　差压传感器的原理（a）和流量与差压的关系（b）

5　产品的特点

MEMS 差压传感器（D6F-PH）采用热式质量流量方式（热式），比薄膜式差压传感器能更灵敏地检测出低压区的变化。

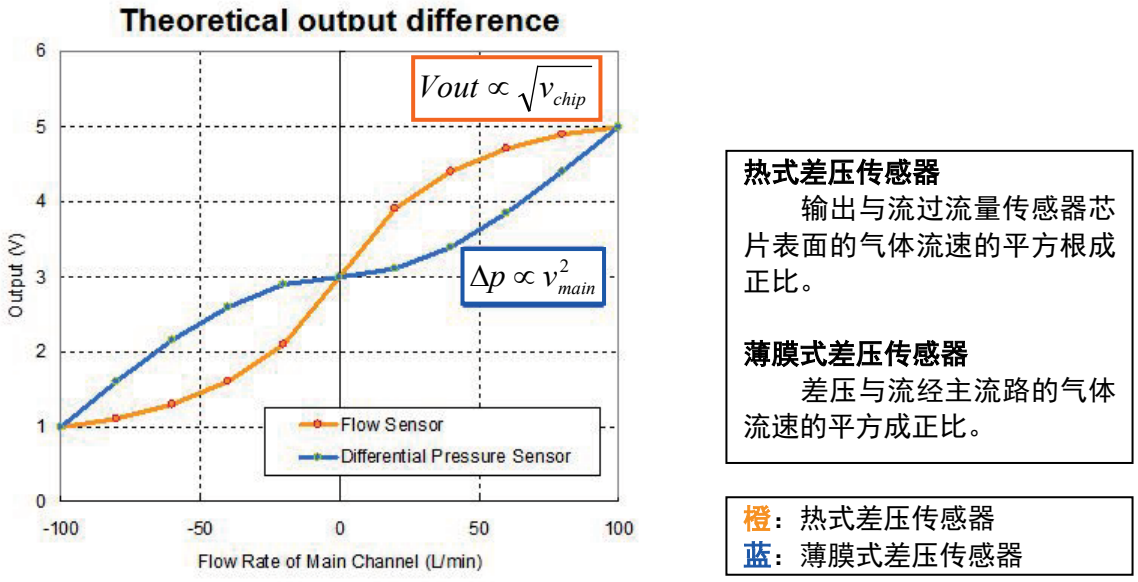


图 6　薄膜式差压传感器与热式差压传感器的比较

表 1　D6F-PH 产品系列

差压范围	接头	连接	型号
0-250Pa	竹节接头	基板封装	D6F-PH0025AD1
		接插件	D6F-PH0025AD2
	歧管	接插件	D6F-PH0025AMD2
±50Pa	竹节接头	基板封装	D6F-PH0505AD3
		接插件	D6F-PH0505AD4
	歧管	接插件	D6F-PH0505AMD4
±500Pa	竹节接头	基板封装	D6F-PH5050AD3
		接插件	D6F-PH5050AD4
	歧管	接插件	D6F-PH5050AMD4

表 2　D6F-PH□□□□的主要规格

项目	内容				
	Min	Typ	Max	单位	备注
差压范围	-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
分辨率	-	12	-	bit	
零点精度（注）	-0.2	-	+0.2	Pa	
量程精度（注）	-3	-	+3	%R.D.	
因温度变动引起的量程移位	-0.5	-	+0.5	%R.D.	相对于 10℃的变化
响应时间	-	33	50	msec	12bit 分辨率
使用环境温度范围	-20	-	80	℃	无结冰，无凝露
储存环境温度范围	-40	-	80	℃	无结冰，无凝露
使用环境湿度范围	35	-	85	%RH	无结冰，无凝露
保存环境湿度范围	35	-	85	%RH	无结冰，无凝露
电源电压	2.3	3.3	3.6	VDC	
消耗电流	-	-	6	mA	Vcc=3.3V、25℃
SCL 频率	-	-	400	kHz	支持 FAST Mode

（注）　零点精度和量程精度是各自独立的误差，不能同时满足。

6 使用方法

6.1 流路的连接方法

通过在作为微小压力变化测量对象的主流路上设置节流孔，使节流孔前后产生微小压力差。从节流孔前后设置的压力端口通过旁路连接到 D6F-PH 上。通过这种连接方式可检测节流孔前后发生的微小压力差。

(1)竹节接头时　(D6F-PH0505AD3 / D6F-PH0025AD1 / D6F-PH5050AD3
D6F-PH0505AD4 / D6F-PH0025AD2 / D6F-PH5050AD4)

与 D6F-PH 连接时使用的配管的内径应为 4.0[mm]，从主流路到传感器间的旁通流路长度应在 800[mm]以下。请尽可能确保配管笔直。

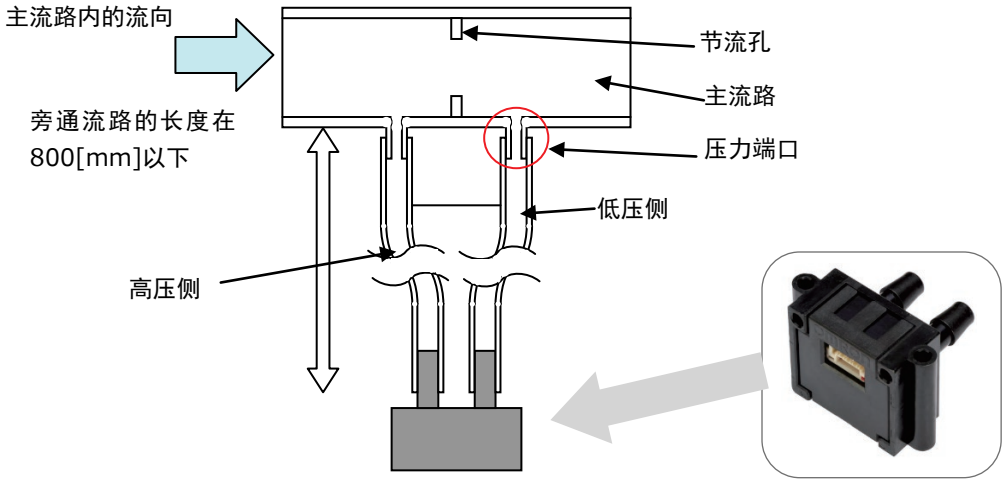


图 7　D6F-PH（竹节接头）主流路的连接

(2) 歧管时　(D6F-PH0505AMD4 / D6F-PH0025AMD2 / D6F-PH5050AMD4)

安装与 D6F-PH 连接的部分时必须用 O 型圈等密封。

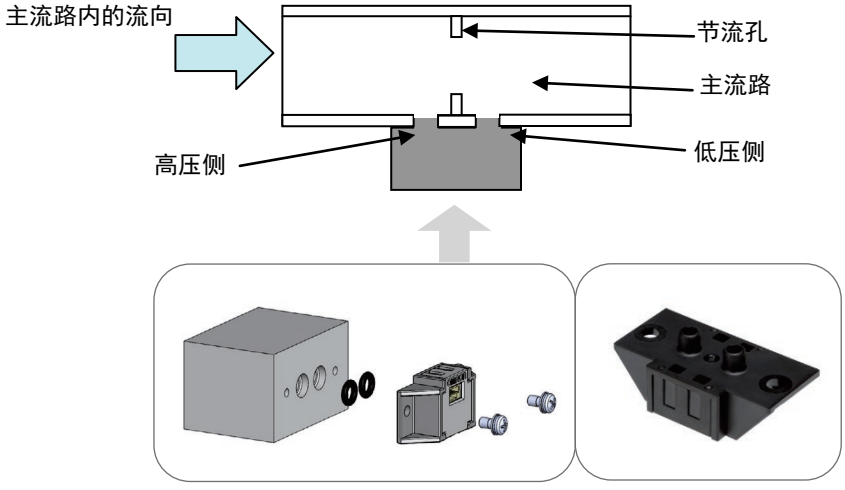


图 8　D6F-PH（歧管接头）主流路的连接

6.2 旁路设计

D6F-PH 的连接流路请按照以下步骤进行设计和评估。以下所示的数值为理论值，实际请通过用户的系统评估后确定。

[STEP1]
请确定下列必备要素。

- 流经系统中主流路的最大流量 ： Fmax
- 压力范围（旁路部分容许的压损） ： Pmax
- 主流路的内径 ： D

Fmax 取决于所用泵的性能。Fmax 时旁路压损为 Pmax。

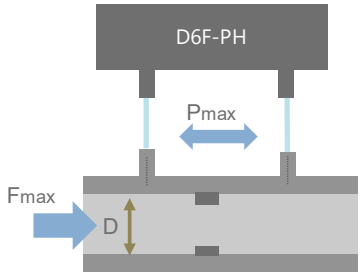


图 9　旁路结构

[STEP2]
D6F-PH 的型号取决于压力范围 Pmax。请选择 D6F-PH。

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

[STEP3]
请参照下表，根据 Fmax 和 D 确定节流孔径 d (mm)。

表 3　节流孔径 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　节流孔径 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　节流孔径 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）对泵进行控制，使流经主流路的最大流量为 100 l/min。此时，若旁路部分的压损为 500 Pa，则主流路的内径 D 为 30mm 时，节流孔径 d 为 9.6mm。（参照表 4）

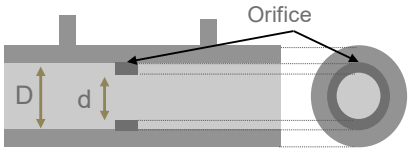


图 10　节流孔

[STEP4]
请利用 STEP3 中确定的节流孔径 d，进行流路的设计和制作。
a = 5 mm
b = 5 mm
c = 1～2mm 左右
L = 10 D 以上
（助跑距离 L 应为笔直距离。助跑距离不足时，请将 L 设为左右对称的值。）

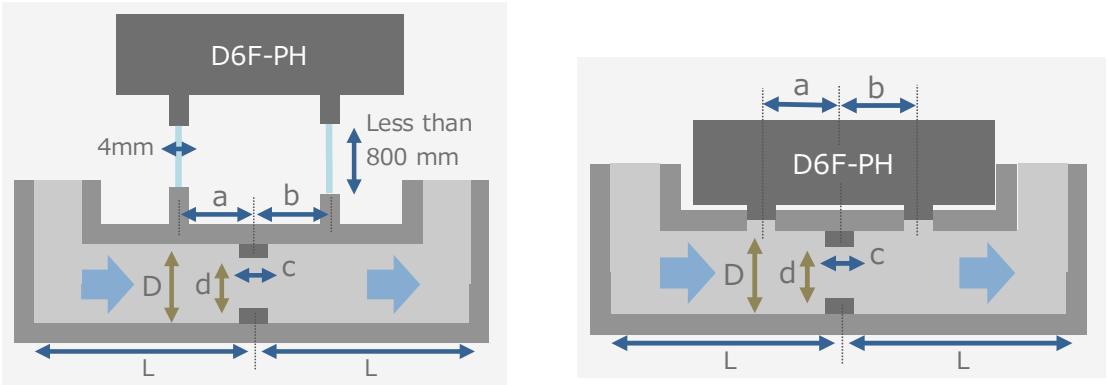


图 11　流路结构示例（左：竹节接头、右：歧管接头）

[STEP5]
请使用设计和制作的流路进行实测，确认 D6F-PH 是否输出正确的压力值。
未能输出正确值时，请变更流路设计的尺寸值，调整为正确值。

（例）
确认流量 Fmax 时 D6F-PH 的输出值。
（理想情况下，流量为 Fmax 时，D6F-PH 的压力输出值接近 Pmax。）
如果 D6F-PH 的压力值存在用户系统不容许的偏差，则调整节流孔径 d 和主流路内径 D 的比率。
・D6F-PH 的输出大于 Pmax 时，增大 d/D 的值。
・D6F-PH 的输出小于 Pmax 时，减小 d/D 的值。
重新制作流路，再次实施评估。

6.3 D6F-PH 的电气连接方法

为了实现 I2C 输出，D6F-PH 必须在数据线（SDA）和时钟线（SCL）上分别设置上拉电阻。如图 6 所示，在与 Vcc 之间安装上拉电阻 2.2[kΩ](推荐值)。
另外，使用时请根据使用的配线长度等，将上拉电阻和 SCL 的传送速度变更为适当的值。

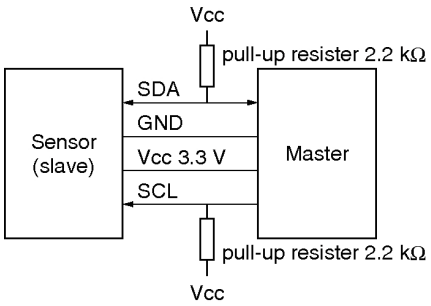


图 12　D6F-PH 的电气连接方法

※ 关于流量传感器连接的注意事项
用户使用环境中的干扰影响可能会导致通信时发生错误。此时，请确认以下几点，需要改善通信错误。

①通信速度的确认
本产品最大支持400 kHz的SCL频率，但在容易发生通信错误的情况下，建议将SCL频率设为低频率后使用。

②配线电缆的确认
连接用户控制的微控制器和本公司流量传感器的电缆较长时，会更容易受到干扰影响。这种情况下建议使用屏蔽电缆。

③上拉电阻值的确认
本产品的I2C通信需要上拉电阻。推荐电阻值为2.2 [kΩ]，请根据客户控制的微控制器和本公司流量传感器之间的连接电缆长度选择最合适的电阻值。另外，如果传感器侧没有回复ACK，请判断为通信异常。ACK回复时间相当于SCL的1 clock周期，因此如果超出该时间仍没有回复ACK，则为通信异常。在这种情况下，必须暂时关闭电源。

7 通信规格

7.1 I2C 接口概述

表 6 I2C 通信基本规格

型号		D6F-PH0025AD1 D6F-PH0505AD3 D6F-PH5050AD3 D6F-PH0025AD2 D6F-PH0505AD4 D6F-PH5050AD4 D6F-PH0025AMD2 D6F-PH0505AMD4 D6F-PH5050AMD4	D6F-PH0025AD1-1 D6F-PH0505AD3-1 D6F-PH5050AD3-1	D6F-PH0025AD1-2 D6F-PH0505AD3-2 D6F-PH5050AD3-2	D6F-PH0025AD1-3 D6F-PH0505AD3-3 D6F-PH5050AD3-3
通信方式		I2C			
从站地址	HEX	0x6C	0x6D	0x6E	0x6F
	BIN(7bit)	110_1100	110_1101	110_1110	110_1111
通信频率		Max. 400kHz			
信号	SCL	Serial Clock			
	SDA	Data Signal			

表 7 I2C 从站地址如下表述。（0x6C 的示例）

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

Write 时：将从站地址的LSB设置为“0”，则为 D8h (1101_1000b)。

Read 时：将从站地址的LSB设置为“1”，则为 D9h (1101_1001b)。

7.2 动作步骤

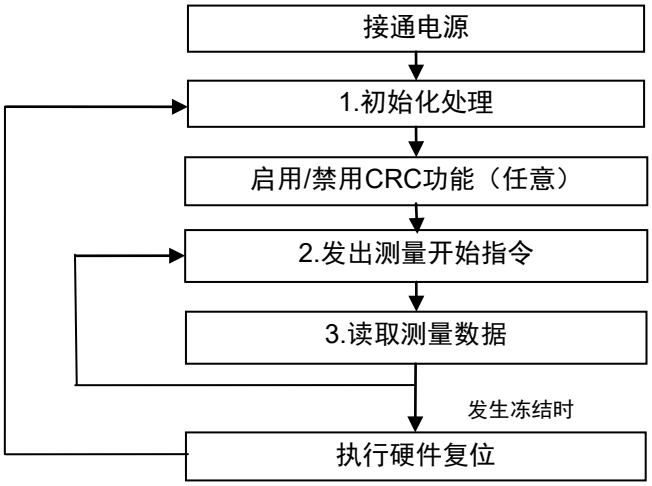


图 13　传感器工作时的流程图

・关于通信时间

项目	符号	备注栏
响应时间	α	α ⩾ 33 ms
采样间隔	β	β > α

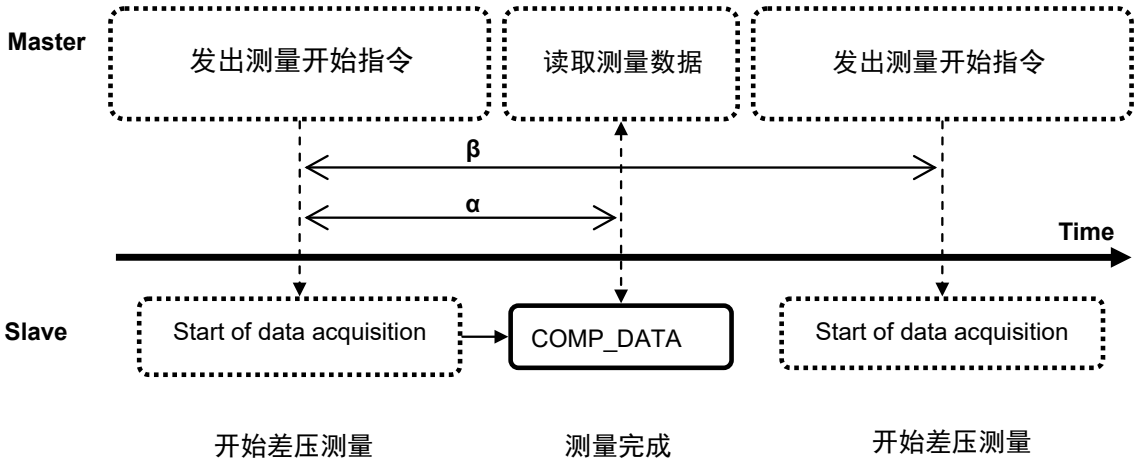


图 14　差压测量的时间轴示意图

1.初始化处理

请在电源接通200μs后实施。

I2C 指令：将 00h 写入初始化寄存器(0Bh)。

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

2.发出测量开始指令

与各种设定一起执行MCU模式（实施差压测量）。写入后读取该寄存器，即可读取MCU选择的MUX的状态。执行处理之后，MS位被设置为“0”。

MCU动作中请不要访问设备。请在经过33msec后访问。

I2C 指令：通过访问地址寄存器(00h)向传感器控制寄存器（D040h）中写入06h(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.读取测量数据

初始化处理后第一个数据不是正常数据，请忽略。

I2C 指令：通过访问地址寄存器(00h)写入修正流量值数据寄存器（D051h、D052h）的读取指令2Ch（2字节读取访问）。

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 指令：通过读缓冲区寄存器（07h、08h）读取流量数据2字节。

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

读取的 16bit 数据为无符号数据。将其作为 Pv，代入下列公式转换为差压(Pa)。

○ 差压测量范围　±50[Pa]　机型时

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

○ 差压测量范围　±500[Pa]　机型时

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

○ 差压测量范围　0-250[Pa]　机型时

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

4.CRC 运算功能的启用（任意）

将CRC运算控制寄存器的位1设置为“1”。CRC值读取时序参照7.4。

I2C 指令：通过访问地址寄存器(00h)向CRC运算控制寄存器（D049h）中写入02h(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.执行硬件复位

将电源时序寄存器的位7设置为“1”。请注意，执行硬件复位后，电源时序寄存器中的位7将自动清除为“0”。

I2C 指令：向电源时序寄存器（0Dh）中写入 80h（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 寄存器详细说明

D6F-PH 通过配置寄存器执行通信。

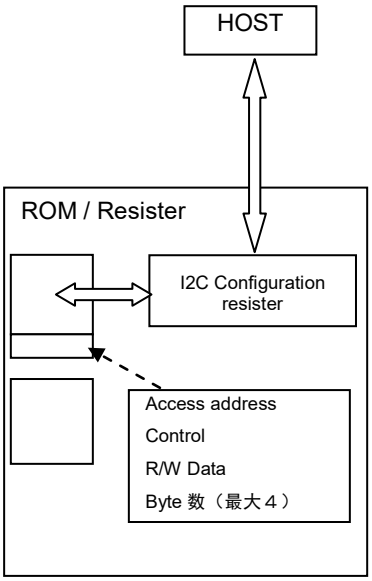


图 15　配置概要

表 8　配置寄存器一览表

Configuration 地址	寄存器名	补充说明
00h	访问地址 1（高位字节）	第一个访问地址的高位字节
01h	访问地址 2（低位字节）	第一个访问地址的低位字节
02h	串行控制	读写访问控制
03h	写缓冲区 0	写入访问地址的数据
04h	写缓冲区 1	写入访问地址+ 1 的数据
05h	写缓冲区 2	写入访问地址+ 2 的数据
06h	写缓冲区 3	写入访问地址+ 3 的数据
07h	读缓冲区 0	从访问地址读取的数据
08h	读缓冲区 1	从访问地址+1 读取的数据
09h	读缓冲区 2	从访问地址+2 读取的数据
0Ah	读缓冲区 3	从访问地址+3 读取的数据
0Bh	初始化寄存器	电源接通后立即进行的初始化控制
0Dh	电源时序	硬件复位功能控制

高位字节：16bit 数据的 bit[15:8]的 1 字节、低位字节：16bit 数据的 bit[7:0]的 1 字节

7.3.1 访问地址寄存器（00h – 01h)

访问地址寄存器用于访问内部寄存器。此寄存器在传输多个字节时自动递增地址，并指定开始地址。

表 9　访问地址寄存器

地址	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

内部寄存器有以下几种。内部寄存器的地址由 16 bit 构成，将该地址写入访问地址寄存器(00h 和 01h)即可进行访问。

表 10　内部寄存器一览表

地址	寄存器名	寄存器说明
D040h	SENS_CTRL	传感器控制寄存器
D046h	FLAGS	标志寄存器
D049h	INT_CTRL	CRC 运算控制寄存器
D051h	COMP_DATA1_H	修正流量值寄存器
D052h	COMP_DATA1_L	
D061h	TMP_H	内部温度传感器值寄存器
D062h	TMP_L	

表 11　SENS_CTRL　(D040h)：传感器控制寄存器

地址	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] – 主设备电源模式设定

0 0 = 待机 － 所有块都处于断电状态。

1 0 = MCU ON － MCU 块工作时需要设定。模拟部、存储器部的电源变为接通状态，开始向 MCU 提供时钟。请不要在测量过程中更改此寄存器的状态。

MS – MCU 启动 – 根据 DV_PWR 的状态执行 MCU 模式。

0 = 停止 - 测量处理时序停止，各块变为断电状态。

1 = 开始 - 开始提供 MCU 时钟，根据设定执行 MCU 模式。

表 12　FLAGS (D046h)：标志寄存器

地址	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 – 供给电压(VDD) 标志：　0 = 供给电压在规格范围内。　1 = 供给电压超出规格范围。

HV1 – 加热器供给电压标志：　0 = 加热器供给电压在规格范围内。　1 = 加热器供给电压超出规格范围。

OS1 – 传感器连接检测标志：　0 = 传感器已连接。　1 = 传感器未连接。

*访问其他位时必须设定为“0”。读取标记寄存器时，为了避免与 MCU 更新冲突，建议读取两次。

表 13　INT_CTRL (D049h)：CRC 运算控制寄存器

地址	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 校验功能选择　（CRC 只支持 2 字节读取。CRC 运算的详情请参照 7.4。）

0 = 禁用 CRC 校验运算功能　　1 = 启用 CRC 校验运算功能

表 14　修正流量值寄存器（D051h、D052h），内部温度传感器值寄存器（D061h、D062h）

地址	寄存器名称	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	说明
D051h	COMP_DATA1_H	DATA<15:8>								修正后差压数据
D052h	COMP_DATA1_L	DATA<7:0>								
D061h	TMP_H	DATA<15:8>								内部温度传感器数据
D062h	TMP_L	DATA<7:0>								

・COMP_DATA1_H 和 COMP_DATA1_L [D051h – D052h]：修正后差压数据（无符号：Uint16）

○ ±50[Pa]、±500[Pa]机型时

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

○ 0-250[Pa]机型时

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

・TMP_H和TMP_L [D061h – D062h]：ASIC内部温度传感器数据（带符号：Int16）

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

注：温度数据只作为参考值使用。温度精度在规格上不予保证。

7.3.2 串行控制寄存器（02h)

表 15　串行控制寄存器（02h)

地址	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] – 访问控制位:
  0 0 = 16 位地址访问(A15-A0)　(内部 ROM 及寄存器等)
  0 1 = 8 位地址访问(A7-A0)　 MCU 内部 256 字节 Dual 端口 RAM 访问
  1 0 = reserved
  1 1 = reserved
- R_WZ – 读/ 写 选择位
  0 = 写访问
  1 = 读访问
- Req- 请求位
  0 = 上一个访问请求结束
  1 = 新请求 内部的串行总线网桥控制电路完成请求处理后，将此 Req 标志清除为“0”。写访问请求时，内部总线网桥控制电路将写缓冲区中的数据传送到由访问地址指定的寄存器。读访问时，内部总线网桥控制电路将指定地址的数据存储在读缓冲区中。
- D_byte_cnt3 [3:0]
  传送字节数指定 仅支持 1、2、3、4 字节传输。

7.3.3 写缓冲区寄存器（03h – 06h)

用于将值写入内部寄存器的 4 个写缓冲区。可以通过以下两种方式写入。以下是将 data[0]写入内部寄存器的 Address=A[15:0]的寄存器中的示例。下列 18h 表示通过串行控制寄存器(02h)写入 18h(新写入 1 byte）。

（方法 1）

突发写入下列 5byte。

・00h、A[15:8]、A[7:0]、18h、data[0]

*第一个 00h 为访问地址寄存器(00h)。

（方法 2）

以 2byte 为单位按顺序写入。

・00h、A[15:8]
・01h、A[7:0]
・03h、data [0]
・02h、18h

*当读取串行控制寄存器(02h)时，如果 bit[3]为“0”，则表示写访问完成。

7.3.4 读缓冲区寄存器 (07h – 0Ah)

用于读取内部寄存器值的 4 个读缓冲区。可以通过以下两种方式读取。以下是对内部寄存器的 Address=A[15:0]的寄存器的值读取 1byte 的示例。下列 1Ch 表示通过串行控制寄存器(02h)写入 1Ch(新读取 1 byte)。

（方法 1）

突发写入下列 4byte，发出读取请求。

・00h、A[15:8]、A[7:0]、1Ch

读请求处理完成后，读取读缓冲区 07h。

*读请求处理完成后，串行控制寄存器(02h)的请求位将清除为“0”。

（方法 2）

以 2byte 为单位按顺序写入。

・00h、A[15:8]

・01h、A[7:0]

・02h、1Ch

读请求处理完成后，读取读缓冲区 07h。

7.3.5 初始化寄存器 (0Bh)

接通电源后，需要将 00h 写入初始化寄存器(0Bh)。（为了载入 NVM 的修整数据）

7.3.6 电源时序寄存器 (0Dh)

表 16　电源时序寄存器 (0Dh)

地址	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. 电源时序寄存器详情

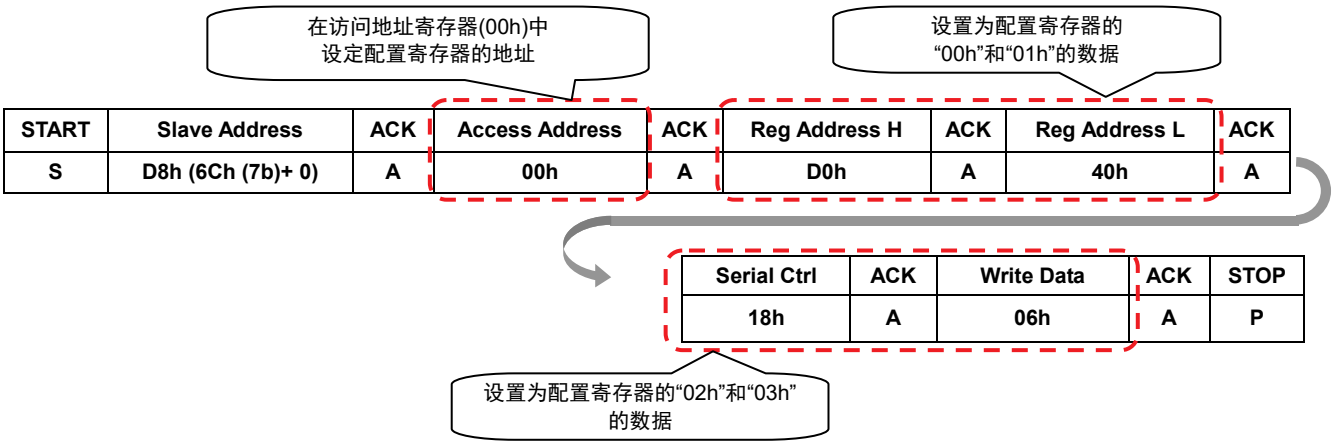
Bit	Name	R/W	Description
[3:0]	Pwr_seq_state5	R	表示电源时序的 state 状态。 h0(0000b): Idle（初始化处理后或电源复位时） h2(0010b): Active（将 06h 写入 D040h 后） h9(1001b): Execute（运算中）
[6:4]	ADC_state	R	控制 ADC 的 state 状态
[7]	Hard_Reset	R/W	1->执行硬件复位（执行后自动清除） 0->不执行硬件复位。

执行硬件复位后，则硬件复位位在执行复位之后自动清除为“0”，内部寄存器返回默认值，内部修整值从非易失性内存重新加载。此硬件复位功能与电源复位功能相同。

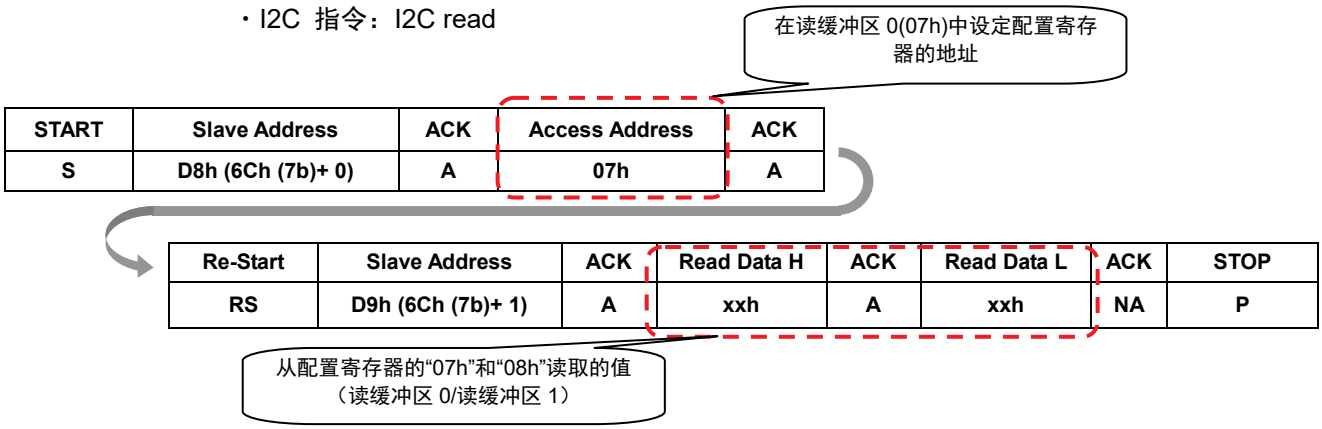
使用硬件复位时，请将 0～6 位作为 0。

7.3.7 I2C 访问指令示例

・I2C 指令：I2C write



・I2C 指令：I2C read



7.4 CRC

・CRC 概要

CRC 被作为数据通信中的错误检测方法使用。本公司的流量传感器使用的是 CRC8 多项式 x^8 + x^5 + x^4 + 1。以下是启用 CRC 功能时的 2 字节读取的 I2C 访问示例。

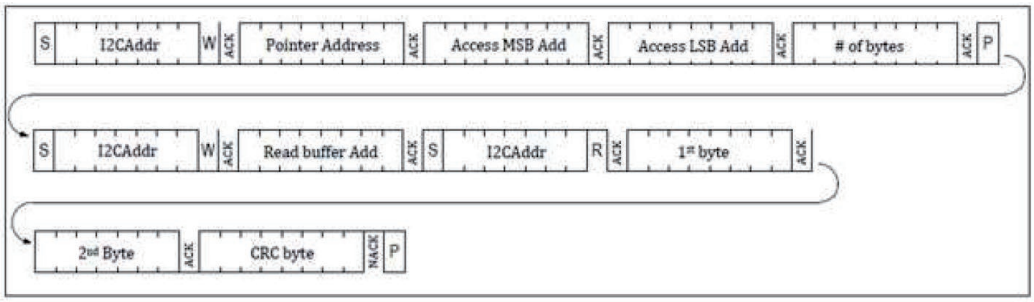


图 16　启用 CRC 功能时的 2 字节数据 + CRC 字节读取示例

・位单位 CRC-8 计算方法

1. 将数据位串排成一列。
2. 将多项式位串排列在数据位串的下方。
3. 如果数据位串最左侧的位为“0”，则将多项式位串向右移动 1 位。如果数据位串最左侧的位为“1”，则与多项式位串一起执行 XOR 运算。然后将多项式位串向右移动 1 位。
4. 重复上述 1.～3.的步骤，直到多项式位串到达数据位串的右端。

以下是基于 XOR 运算的 CRC 字节计算示例。

	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

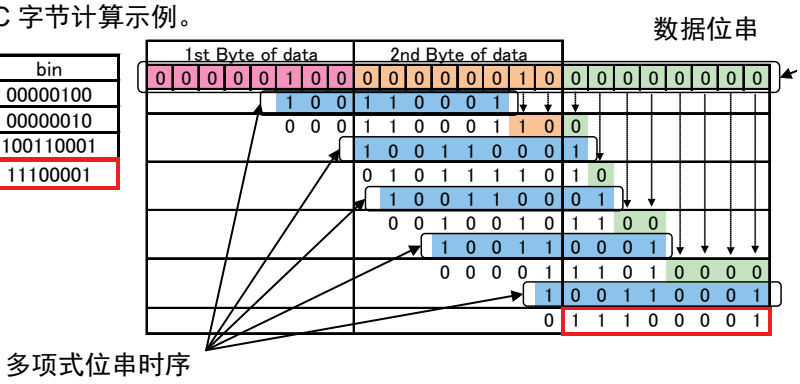


图 17　CRC-8 的 XOR 运算示例

8　开发工具

备有 3 种示例代码，作为软件开发支援工具。

1. Raspberry Pi 用　示例代码
2. Arduino 用　示例代码
3. STM32 MCU 用　示例代码

Raspberry Pi 用示例代码和 Arduino 用示例代码可以与欧姆龙评估板一起使用。欧姆龙评估板支持以下 3 种平台，将差压传感器 D6F-PH、评估板、线束连接至平台，即可轻松评估。
评估板支持所有的 D6F-PH，评估板与 D6F-PH 的连接线束因 D6F-PH 的型号而异，请予以注意。
评估板 URL：(http://www.omron.co.jp/ecb/sensor/evaluation-board/2jcie)

表 18　评估板一览表

平台	评估板	示例代码	连接用线束（评估板与 D6F-PH 之间）	线束对应差压传感器型号
Raspberry Pi *1 连接用	2JCIE-EV01-RP1 	https://github.com/omron-devhub/d6f-2jcieev01-raspberrypi	2JCIE-HARNESS-02 *4	D6F-PH0025AD1
				D6F-PH0505AD3
				D6F-PH5050AD3
Arduino *2 连接用	2JCIE-EV01-AR1 	https://github.com/omron-devhub/d6f-2jcieev01-arduino	2JCIE-HARNESS-03 *5	D6F-PH0025AD2
				D6F-PH0505AD4
				D6F-PH5050AD4
ESP32 Feather *3 连接用	2JCIE-EV01-FT1 	https://github.com/omron-devhub/d6f-2jcieev01-arduino		D6F-PH0025AMD2
				D6F-PH0505AMD4
				D6F-PH5050AMD4

*1. Raspberry Pi 是 Raspberry Pi 财团的注册商标。
*2. Arduino 是 Arduino LLC 和 Arduino SRL 的注册商标。
*3. Feather 是 Adafruit Industries LLC 的注册商标。
*4. 2JCIE-HARNESS-02 的一侧为接插件，另一侧为引线。需要将引线与 D6F-PH 连接起来使用。
*5. 2JCIE-HARNESS-03 的两侧均为接插件。D6F-PH 与评估板可通过接插件简单连接。

即使不使用评估板，也可利用示例代码。但需由用户对传感器进行接线。此外，示例代码用于评估，欧姆龙不保证正常运行。

8.1 Raspberry Pi 用示例代码

Raspberry Pi 用示例代码在以下 URL 中。

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

--差压测量范围　0～250Pa 用
https://github.com/omron-devhub/d6f-2jcieev01-raspberrypi/blob/master/d6f-ph0025.c

--差压测量范围　±50Pa 用
https://github.com/omron-devhub/d6f-2jcieev01-raspberrypi/blob/master/d6f-ph0505.c

--差压测量范围　±500Pa 用
https://github.com/omron-devhub/d6f-2jcieev01-raspberrypi/blob/master/d6f-ph5050.c

示例代码的结构如下所示。

```
/** <!-- 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");
    }
}
```

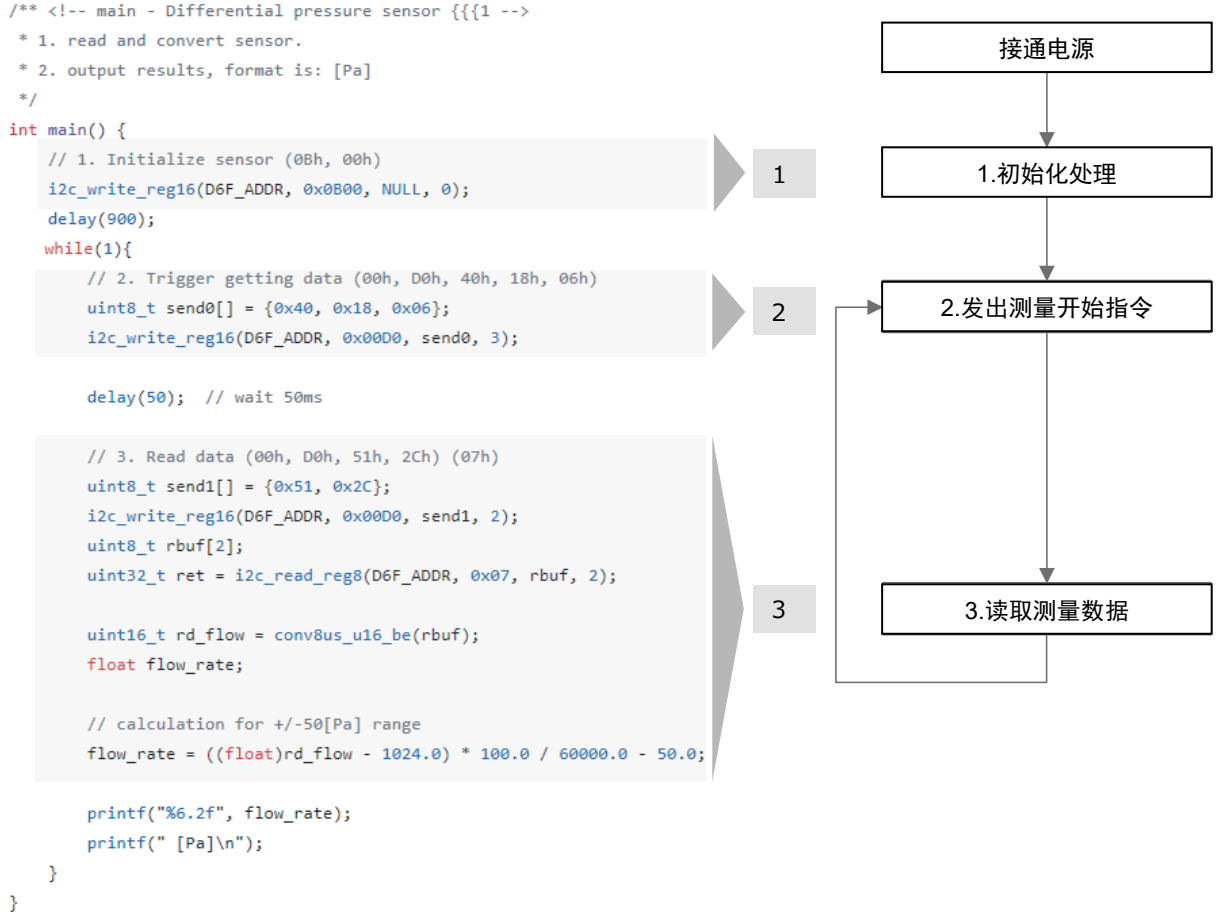


图 18　Raspberry Pi 用示例代码结构

Raspberry Pi 用示例代码的运行顺序如下所示。

(1)将 D6F-PH、线束、欧姆龙评估板(2JCIE-EV01-RP1)与 Raspberry Pi 连接

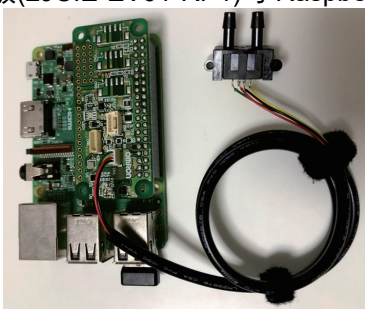


图 19　Set-up

(2)启用 I2C

启动 Raspberry Pi，从 Start 菜单中打开“Preferences”>“Raspberry Pi Configuration”，将 I2C 设定为“Enable”后重启。

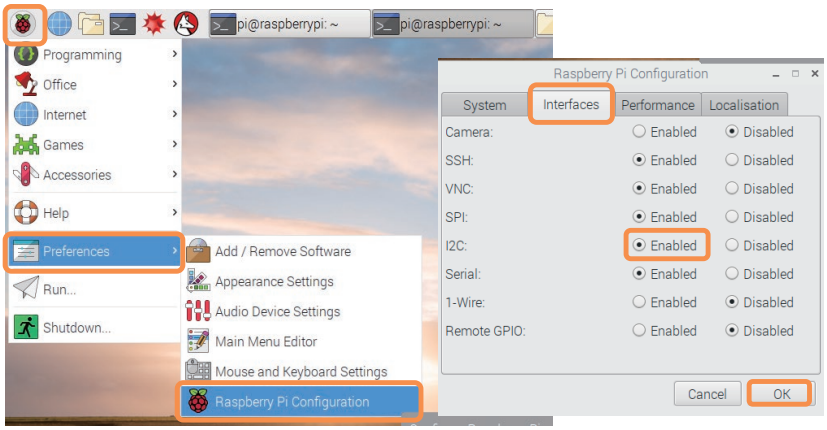


图 20　Enable I2C

(3)下载示例代码

从以下 URL 访问 GitHub，下载 Zipfile。

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

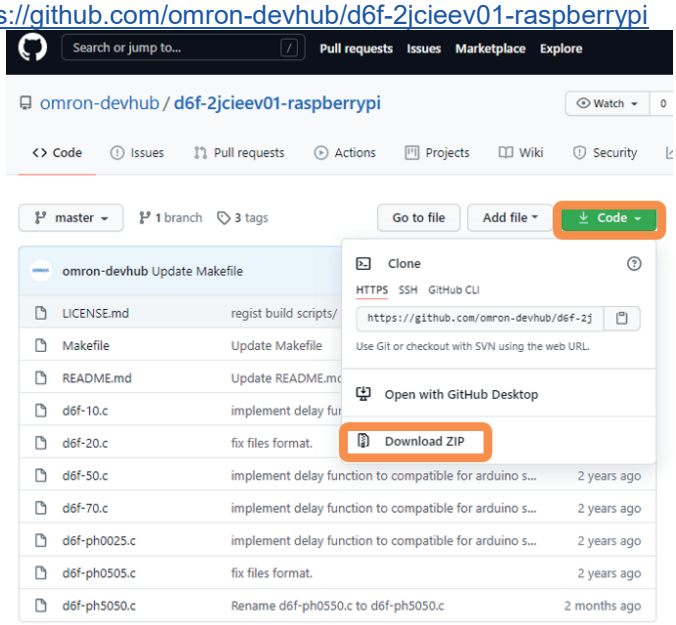


图 21　Download sample code

或者，打开 Terminal，执行以下指令。

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

或者，通过连接英特网的其他 PC 下载 Zip file 后，用 USB 存储器等将 zip file 转移到 Raspberry Pi 中。

(4) Make file

打开 Terminal，执行以下指令。

```
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
```

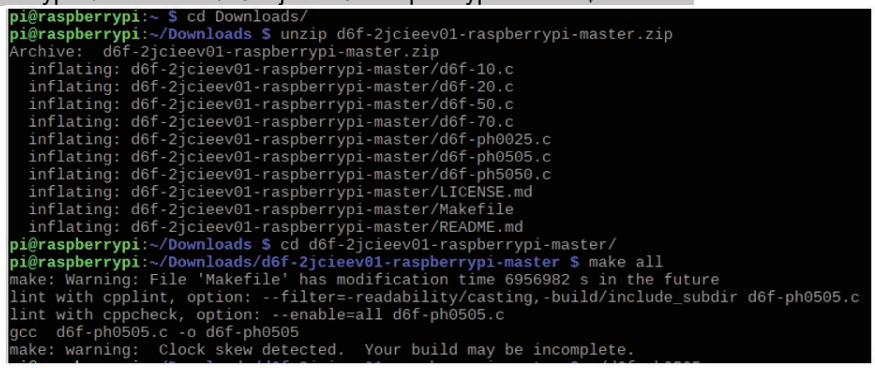


图 22　Make file

(5) Run the file

为了获取数据，执行以下指令。

--差压测量范围　0～250Pa 时

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

--差压测量范围　±50Pa 时

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

--差压测量范围　±500Pa 时

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

（需停止获取数据时，请同时按下“Ctrl”键和“C”键。）

图 23　Run the file

8.2 Arduino 用示例代码

Arduino 用示例代码在以下 URL 中。

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

--差压测量范围　0～250Pa 用
https://github.com/omron-devhub/d6f-2jcieev01-arduino/tree/master/examples/d6f-ph0025

--差压测量范围　±50Pa 用
https://github.com/omron-devhub/d6f-2jcieev01-arduino/tree/master/examples/d6f-ph0505

--差压测量范围　±500Pa 用
https://github.com/omron-devhub/d6f-2jcieev01-arduino/tree/master/examples/d6f-ph5050

示例代码的结构如下所示。

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

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

    // 1. Initialize sensor (08h, 00h)
    i2c_write_reg16(D6F_ADDR, 0x0800, 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]");
}
```

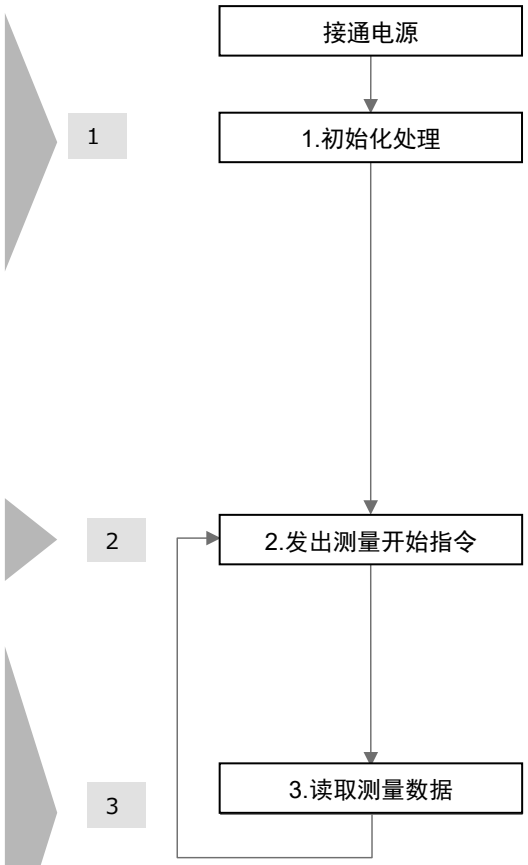


图 24　Arduino 用示例代码结构

Arduino 用示例代码的运行顺序如下所示。

(1)将 D6F-PH、线束、欧姆龙评估板(2JCIE-EV01-AR1)与 Arduino 连接

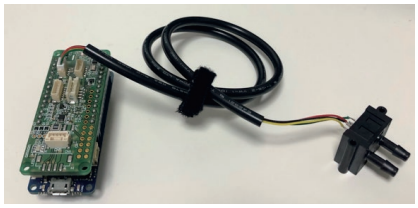


图 25　Set up

(2)下载 Arduino IDE

从以下 UARL 下载 Arduino IDE。

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

(3)在 Arduino IDE 上识别 Arduino

打开 Arduino IDE，经由 USB 将 Arduino 连接至 PC。

显示以下内容时，安装 Arduino MKR 用的 Package。

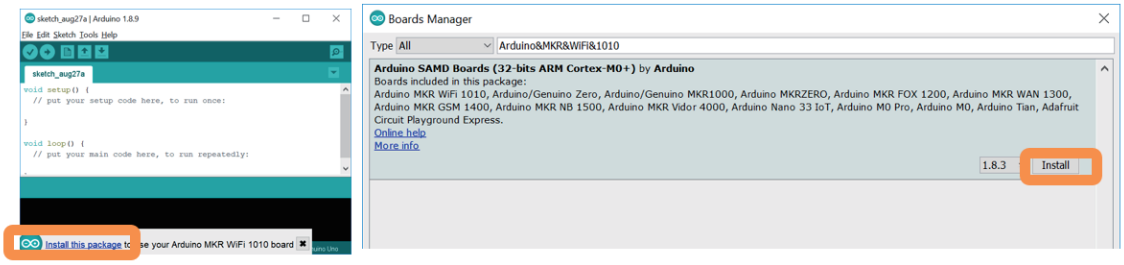


图 26　Arduino IDE

安装 Driver。

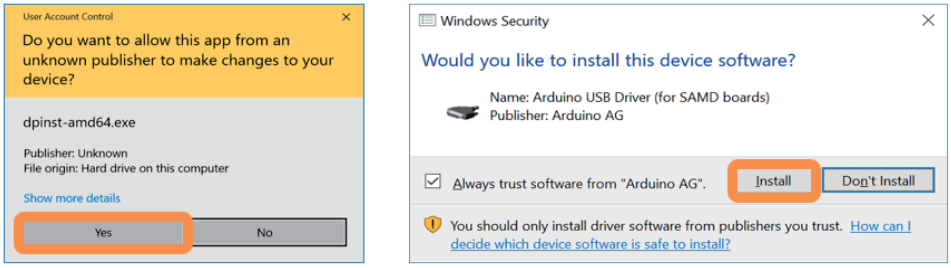


图 27　Driver

在 Windows 的开始菜单中检索“Device Manager”。

确认 PC 识别的 Arduino 的 COM 端口编号。

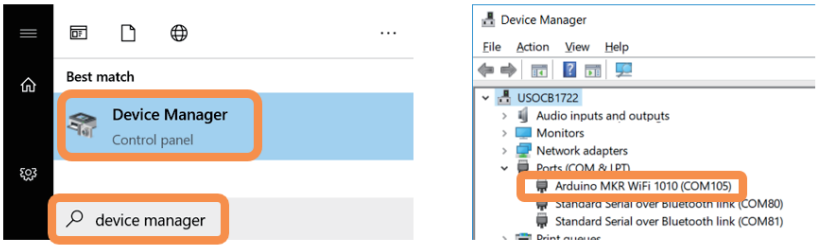


图 28　Device manager

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

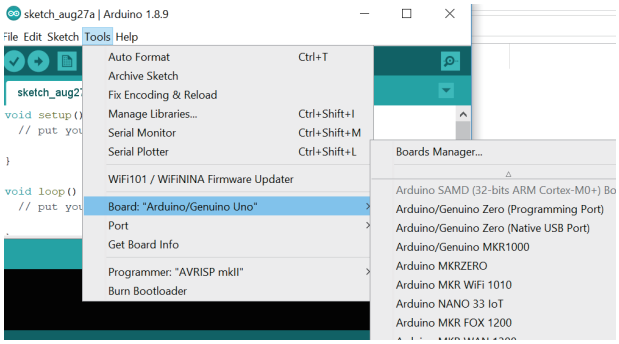


图 29　Select Arduino

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

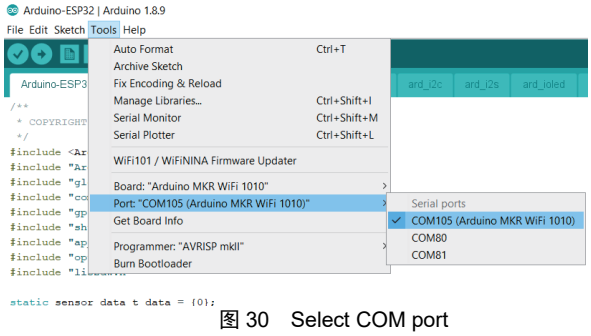


图 30　Select COM port

(4)下载示例代码

连接至以下 GitHub 的 URL，下载 zip file。

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

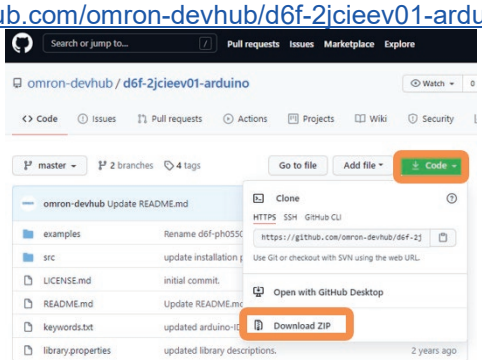


图 31　Download sample code

(5)将示例代码上传到 Arduino。

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

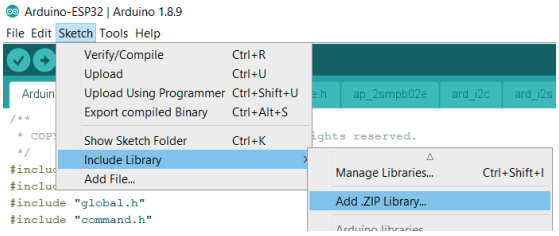


图 32　Add zip

选择“Downloads”文件夹。选择“d6f-2jcieev01-arduino-master.zip”。

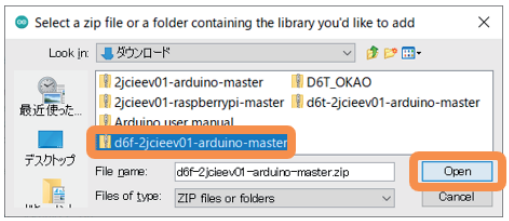


图 33　Select zip file

根据传感器的差压测量范围选择文件。
“File” -> “Examples” -> “D6F-2JCIE-EV01”
-> “d6f-ph0025” or “d6f-ph0505” or “d6f-ph5050”

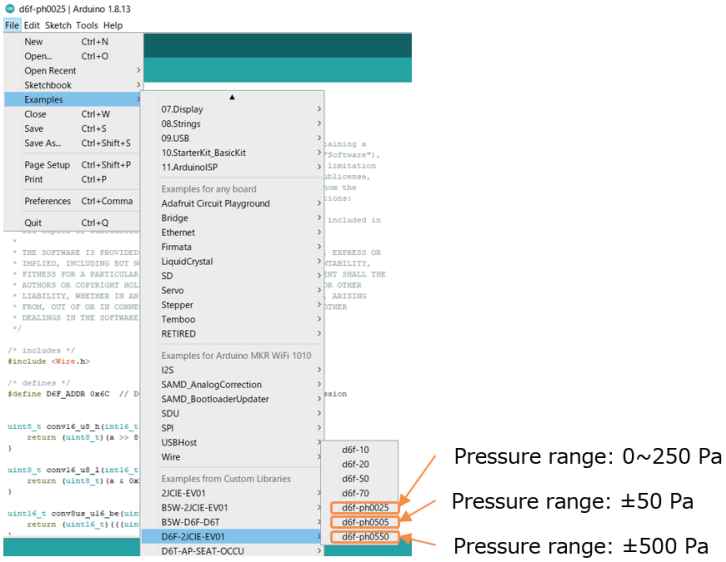


图 34　Select file

点击“Verify”。确认是否出现错误。

图 35　Verify

点击“Upload”。确认是否显示“CPU reset”。

图 36　Upload

(6)获取数据

“Tools” -> “Serial Monitor”

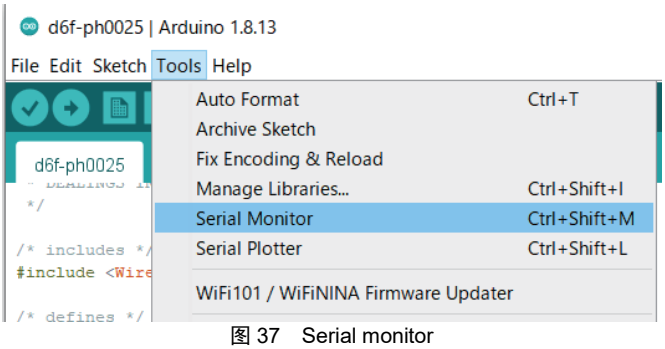


图 37　Serial monitor

显示数据。

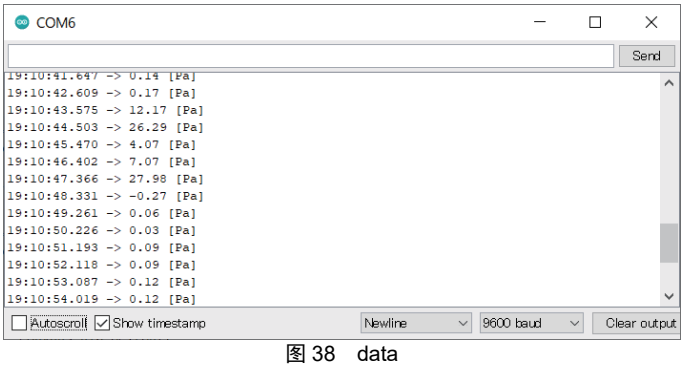


图 38　data

8.3 STM32 MCU 用示例代码

STM32 MCU 用的示例代码为一个实例。需要根据实际使用的 MCU 的规格变更 I2C 控制的函数等。

请根据所用的差压测量范围变更 D6F_PH_Sample.h 的以下部分。

```
/*================================================*/
/* D6F-PH Digital Flow Sensor Header File (using STM32)
 * :Copyright: (C) OMRON Corporation, Microdevice H.Q.
* 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);
```

Pressure range: ±50 Pa

```
#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]
```

Pressure range: 0~250 Pa

```
//#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]
```

Pressure range: ±500 Pa

```
//#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]
```

图 39　header file

D6F_PH_Sample.c 的示例代码结构如下所示。

```
/*================================================*/
/* 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;
}
```

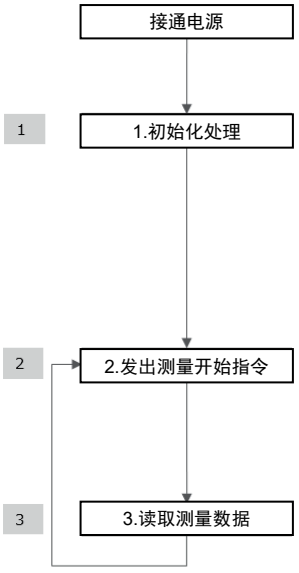


图 40　main code

该示例代码将从传感器输出的 16bit 无符号整数的值转换为 16 位带符号整数（2 的补数）乘以 10 倍的压力值[Pa]。

例如，16 位带符号整数（2 的补数）乘以 10 倍的压力值[Pa]rd_flow 为 218 时，表示 21.8Pa。

示例代码

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 承诺事项

首先真诚地感谢您一直以来对欧姆龙株式会社（以下简称“本公司”）产品的支持。
关于“本公司产品”的购买，无论客户从何处购买，均适用本承诺事项中的条件。请在同意的基础上进行订购。

1. 定义

本承诺事项中术语的定义如下所示。
(1)“本公司产品”：“本公司”的 FA 系统设备、通用控制设备、传感设备、电子和机械零件
(2)“产品样本等”：与“本公司产品”相关的欧姆龙综合样本、FA 系统设备综合样本、安全元器件综合样本、电子和机械零件综合样本、其它产品样本、规格书、使用说明书、手册等，还包括通过电磁介质提供的资料。
(3)“使用条件等”：“产品样本等”中的“本公司产品”的使用条件、额定值、性能、运行环境、使用方法、使用注意事项、禁止事项等
(4)“用户用途”：用户使用“本公司产品”的方法，包括直接使用或将“本公司产品”装入用户制造的零件、印刷电路板、机械、设备或系统等。
(5)“适用性等”：“用户用途”中“本公司产品”的（a）适用性、（b）动作、（c）不侵犯第三方知识产权、（d）遵守法律以及（e）遵守各种标准

2. 记载内容的注意事项

关于“产品样本等”中的内容，请注意以下几点。
(1)额定值和性能值是在各条件下进行单独试验后获取的值，并不保证在复合条件下可获取各额定值和性能值。
(2)参考数据仅供参考，并不保证始终在该范围内正常运行。
(3)使用实例仅供参考，“本公司”不保证“适用性等”。
(4)“本公司”可能会因产品改良、本公司的原因而中止“本公司产品”的生产或变更“本公司产品”的规格。

3. 使用注意事项

使用时，请注意以下几点。
(1)使用时请符合额定值、性能以及“使用条件等”。
(2)请用户自行确认“适用性等”，判断是否可使用“本公司产品”。“本公司”对“适用性等”不作任何保证。
(3)用户将“本公司产品”用于整个系统时，请务必事先自行确认配电、设置是否恰当。
(4)使用“本公司产品”时，请注意以下各事项。(i)使用“本公司产品”时，应在额定值和性能方面留有余量，(ii)采用冗余设计等安全设计，即使“本公司产品”发生故障，也可将“用户用途”造成的危险降至最低程度，(iii)对整个系统采取向使用者告知危险的安全措施，(iv)定期维护“本公司产品”及“用户用途”。
(5)本公司设计并制造面向一般工业产品的通用产品。因此，不可用于以下用途。如果用户将“本公司产品”用于以下用途，则“本公司”不对“本公司产品”作任何保证。并且，即使是起重设备、医疗设备以下列举的用途，根据其具体的使用方法，可以作为面向一般工业产品的通用产品提供常规的保修，请咨询本公司销售负责人。
(a)需高安全性的用途（例：核能控制设备、燃烧设备、航空航天设备、铁路设备、起重设备、娱乐设备、医疗设备、安全装置以及其他危及生命、健康的用途)
(b)需高可靠性的用途（例：煤气、自来水、电力供应系统、24 小时持续运行的系统以及支付系统等涉及权利、财产的用途等）
(c)在严苛条件或环境下使用（例：需设置在室外的设备、会受化学污染的设备、会受电磁干扰的设备、会受振动、冲击的设备等）
(d)在“产品目录等”中未记载的条件或环境下使用
(6)上述 3.(5)(a)～(d)以及“本产品样本等中记载的产品”不可用于汽车（含两轮车。下同）。请勿装入汽车进行使用。关于可装入汽车的产品，请咨询本公司销售负责人。

4. 保修条件
“本产品”的保修条件如下所述。
(1)保修期　为从本公司或本公司代理店购买本产品后的 1 年内。
（“产品样本等”中另有记载的情况除外。）
(2)保修内容　对发生故障的“本公司产品”，经“本公司”判断后提供以下任一服务。
(a)发生故障的“本公司产品”可在本公司维修服务网点免费维修
（不提供电子和机械零件的维修服务。）
(b)免费提供与发生故障的“本公司产品”数量相同的替代品
(3)非保修范围　如果因以下任一原因造成故障，则不在保修范围内。
(a)用于非“本公司产品”原本用途的用途时
(b)未按“使用条件等”进行使用
(c)改造或维修未经“本公司”
(d)使用的软件程序非由“本公司”人员编制
(e)因以出厂时的科学技术水平无法预见的原因
(f)除上述以外，因“本公司”或“本公司产品”以外的原因（包括自然灾害等不可抗力）
5. 责任免除
本承诺事项中的保修即与“本公司产品”相关的保修的所有内容。对因“本公司产品”造成的损害，“本公司”及“本公司产品”的销售店概不负责。
6. 出口管理
出口“本公司产品”或技术资料、或向非居民的人员提供时，应遵守日本及各国安全保障贸易管理相关的法律法规。如果用户违反上述法律法规，则可能无法向其提供“本公司产品”或技术资料。以上

(EC300)

订购前请务必阅读我司网站上的“注意事项”。

欧姆龙电子部品（中国）统辖集团

网站
欧姆龙电子部件贸易（上海）有限公司 https://www.ecb.omron.com.cn

Cat. No. CDSC-CN1-025B 2021年6月