恒定性能PCB泵驱动器：

通信和命令

IDEX HEALTH & SCIENCE

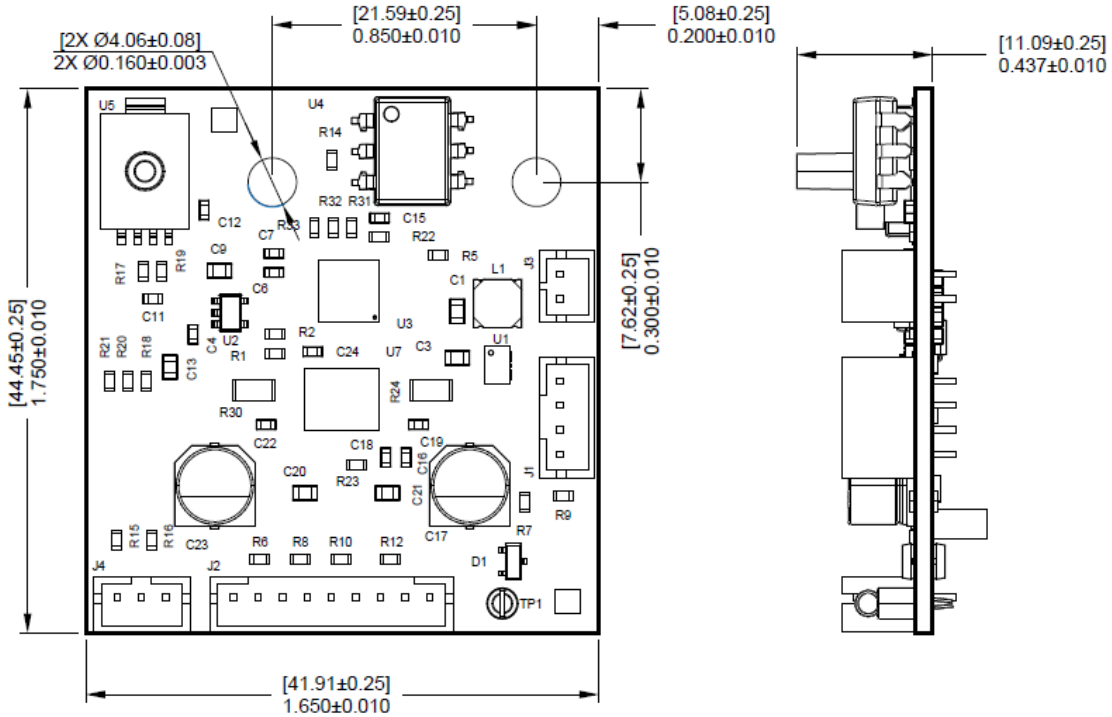
Intelligent Solutions for Life®

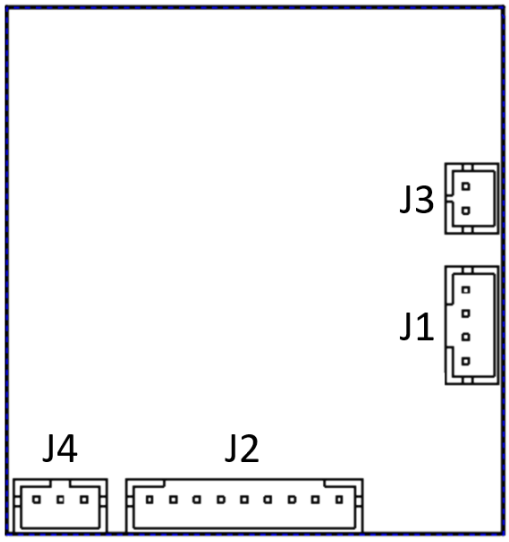
Fluidics | Optics | Consumables | Assemblies

目录

电路板尺寸和连接头位置

尺寸的单位为【毫米】和英寸 仅供参考





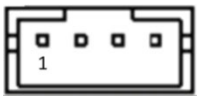
PCB连接

连接头	功能	PCB货号	配合件货号
J1	电源输入	JST B4B-PH-K-S(LF)(SN)	PHR-4
J2	通信 (I²C和UART)	JST B9B-PH-K-S(LF)(SN)	PHR-9
J3	故障输出	JST B2B-PH-K-S(LF)(SN)	PHR-2
J4	LED指示灯	JST B3B-PH-K-S(LF)(SN)	PHR-3

J1 – 电源输入

P1	P2	P3	P4
+24VDC*	+24VDC*	接地	接地

*J1 输入电压变化范围为+15VDC 到+24VDC



J2 –通信 (I²C and UART)

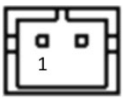
P1	P2	P3	P4	P5	P6	P7	P8	P9
I2C_SDA	I2C_SCL	U1_TX	U1_RX	U0_RX	U0_TX	接地	ISPn	RESETn

I2C_SDA, I2C_SCL:开漏 I2C 引脚

U1_RX, U1_TX:3.3V UART 引脚, 5V 容限

U0_RX, U0_TX, ISPn:仅厂家使用，禁止连接

RESETn:3.3V 逻辑引脚, 5V 容限低信号复位电路板



J3 – 故障输出

P1	P2
发射极	集电极

发射极，集电极：H11G2SR2M 光耦输出引脚

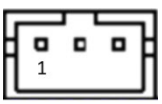
光电隔离输出：
当真空度≤设定值+30mmHg 且未检测到错误时，关闭（不导通）
真空度>设定值+30mmHg 以上或检测到错误时，开启（导通）

当发生错误时，步进电机驱动器关闭，需要重新上电或复位输入的低脉冲来恢复正常操作。

J4 – LED 指示灯

P1	P2	P3
红色LED阳极	接地	绿色LED阳极

琥珀灯闪烁：真空度>设定值+30mmHg
绿灯闪烁：真空度≤设定值+30mmHg，或者真空度小于设定值。
实心绿灯：真空度=设定值
实心红灯：故障（除上述以外的其他状态）
红灯绿灯交替闪烁：向 PCB 发送关闭命令



硬件接口

支持两种硬件接口，I²C 和 UART (最大容限分别为 3.3V, 5V).

接口	点对点或多点	编码	速度 (最大 bits/sec)
I²C	多点	二进制	100K
UART	点对点	ASCII	115.2K

I²C

I²C是多点标准，支持同一总线上的多个单元。各单元可以单独寻址。该实现按照I²C标准进行启动、写入地址、读取地址、确认、否确认和停止条件。
I²C中的地址被编码到第一个字节的高7位。第一个字节的最后一位是读/写位，设置为0表示写入，1表示读取。例如，如果单元地址为9，则I²C的地址字节为18（0001 0010）表示写入，19（0001 0011）表示读取。
I²C的数据包编码是二进制的。

UART

UART是一种点对点标准，只有一个单元可以连接到主机上的UART端口。

除了信令电平外，UART与RS-232相似。

注意 - 不要连接到标准的RS-232端口，以免造成损坏！

UART的电平是标准的3.3V CMOS逻辑电平。 输入阈值为：
V_{IL} = 0.99V (3.3V * .3)
V_{IH} = 2.31V (3.3V * .7)

输入为5V容限，因此可以从0V增加到5V而不导致损坏。
输出电平为：
V_{OL} = 0.4V
V_{OH} = 2.9V

使用IDEX的命令集时会包含地址。地址必须与单元地址匹配，否则该命令将被忽略。

UART 的数据包编码是 ASCII。

命令包格式

命令包的格式为：

发送到设备

地址，长度，命令代码，设备地址，[参数]，CRC

- 地址=4 到 123，0 保留给所有单元广播消息（默认地址为 9）
- 长度=消息的字节计数，包括长度字节和 CRC 字节（不计算地址字节）
- 命令代码=1 字节命令代码（见下表）
- 设备地址=此单元内的子地址（始终为 0）
- 参数=命令所需的可选参数
- CRC=2 字节 CRC-CCITT

设备的响应

命令状态，长度，[数据]，CRC

- 命令状态=0 表示成功状态，>0 表示失败（见下面的命令状态表）
- 长度=消息的字节计数，包括长度字节和 CRC 字节
- 数据=返回值（如果有）
- CRC=2 字节 CRC-CCITT

命令状态：
0–完成命令
4 – 无效 CRC
5 – 无效命令
8 – 参数未知
12 – 缺失开始字符
13 – 错误的数据包大小
14 – 命令超时
15 – 无回车
16 – 非十六进制字符

数据包编码

二进制编码

I²C 接口使用的编码方式很容易理解。数据包（地址、长度、命令、参数、CRC）作为二进制字节发送到 I²C 端口。PCB 是 I²C 从动装置，因此除非主设备要求读取，否则它无法发送响应。如果无需考虑状态，则不必要求读取从动装置。如果想读取状态，则必须发出读取地址，然后按照标准的 I²C 协议进行时钟响应。

以下是 IDEX 命令关闭泵和单元响应的示例（通过 I²C 地址 9）：（灰色是从主机发送的字节，黄色是从单元发送的字节）

0x12	0x06	0x55	0x00	0x00	0x2B	0xD7
地址写入	字节数	命令码	总是 0	值（关闭）	CRC1	CRC2

0x13	0x00	0x03	0x2D	0x6C
地址读取	命令状态	字节数	CRC1	CRC2

以下是 IDEX 命令的示例，此命令设置流速为 5 ml/min，以及该单元的响应：

0x12	0x09	0x7E	0x00	0x00	0x4C	0x4B	0x40	0x77	0xFA
地址写入	长度	命令码	总是 0	值 (5,000,000 nL/min)				CRC1	CRC2

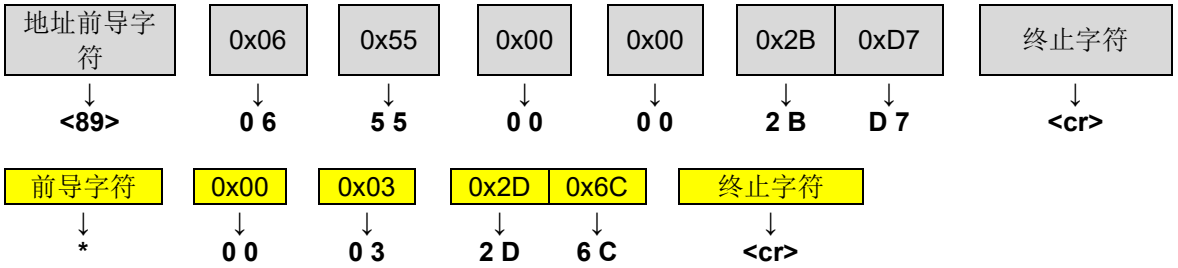
0x13	0x00	0x03	0x2D	0x6C
地址读取	命令状态	字节数	CRC1	CRC2

ASCII编码

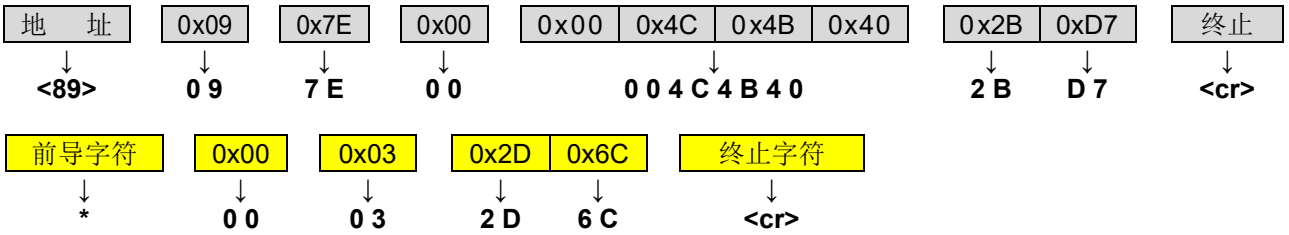
UART接口使用的ASCII编码，在地址编码为ASCII字符串之后发送所有数据。它将地址字节替换为一个前导字节，该字节将地址编码为地址值+128（如下所示为地址9+0x80）。然后，它将消息的其余部分编码为ASCII十六进制数字（“0”到“9”，“A”到“F”）。最后，以回车终止数据包。传输的字符串显示在图示的下方。

由于UART可以独立地接收和发送，因此它将始终发送对收到的任何命令的响应。电路板在U1_RX引脚（J2-引脚4）上接收数据包，并在U1_-TX（J2-引脚3）上发送响应。响应总是以“*”字符（0x2A）开头，以回车结束。

以下是相同的IDEX 关泵命令和以ASCII编码通过UART发送的响应：



以下是IDEX命令的示例，此命令设置流速为5 ml/min，以及该单元的响应：



IDEX命令集

系统命令

命令	代码	参数	返回值
获取供应商名称	0x21	-	'IDEX'
获取固件货号	0x22	-	最多9个ASCII字符，终止为空字符
获取固件版本	0x23	-	2 ASCII 字符: 主要版本，次要版本
获取系统货号	0x24	-	最多9个ASCII字符，终止为空字符
获取系统序列号	0x26	-	最多10个ASCII字符，终止为空字符
获取系统版本	0x29	-	2 ASCII 字符: 主要版本，次要版本
获取制造日期	0x2B	-	制造年 – 2000 制造 月 1 – 12 制造日期 1-31
设置电路板地址	0x2D	1-字节，有效范围从4-123	-
复位	0x2E	-	-
获取命令状态	0x30	-	状态字节
设定波特率	0x33	1 - 9600, 2 – 19200, 3 38400, 4 – 57600, 5 -115200	以当前波特率 返回标准确认数据包，然后切换到 新波特率以用于下一个命令
获取波特率	0x35	-	返回波特率的代码: 1 - 9600, 2 – 19200, 3 – 38400 4 – 57600, 5 -115200
加载默认参数	0x38	-	恢复出厂默认参数
保存参数	0x39	-	将运行参数写入非易失性存储器
获取PCB板货号	0x3A	-	9 ASCII 字符，终止为空字符
获取参数	0x3F	1-字节参数号	4-字节参数值
设定参数	0x40	1-字节参数号 4-字节参数值	-
泵开/关控制	0x55	0 (关闭) 或者1 (打开)	-

命令	代码	参数	返回值
获取真空压力	0x72	-	返回真空压力，单位为毫米汞柱*10
获取状态	0x79	参数1：要读取的int16值的数目。 参数2：表中开始索引 例子：2，1将返回真空值和平均电机速度	返回指定长度的int16数组。 0: 系统状态（0-关闭，1-低压，2-在设定值，3-高压，4-非常高压，5-故障） 1: 真空度 xxx.x mmHg 2: 平均电机速度 xxx.x rpm 3: 脉冲 xxx.x 4: 压力差 xxx.x mmHg 5: 瞬时电机转速 xxx.x rpm 6: PID 故障 xxx.xx mmHg 7: 瞬时真空值 xxx.xx 8: ADC 读数 xxxxx 计数 9: PID比例 xxx.x 10: PID积分 xxx.x
获取PCB板序列号	0x7A	-	最多10个ASCII字符，终止为空字符
获取PCB版本	0x7C	-	2 ASCII 字符: 主要版本，次要版本
设定流速	0x7E	4-字节值 nL/min, 范围为1 到10,000,000 nL/min	-
设置待机	0x80	0–在之前的真空度级别恢复正常操作 1 – 设置真空度级别为288 mmHg	-
设置系统货号	0x25	最多9个ASCII字符，终止为空字符	-
设置系统序列号	0x28	最多10个ASCII字符，终止为空字符	-
设置系统版本	0x2A	2ASCII 字符:主要版本，次要版本	-

设备参数编号

参数由获取参数（0x3F）和设置参数（0x40）命令读取或写入。即使实际参数范围较小，所有参数值都会以32位值（4字节）的形式读取或写入。

参数编号	参数	单位
88 (0x58)	真空度设定值	1/10 mmHg
89 (0x59)	环境大气压力	1/10 mmHg
90 (0x5A)	效率60%到90%	%
94 (0x5E)	泵抽空超时	秒
95 (0x5F)	出错超时	秒

程序示例

下面的代码示例将有助于理解如何开发通信程序。

CRC-CCITT示例

CRC从地址开始计算，一直持续到整个消息。然后将CRC字节附加到消息的末尾，以形成完整的包。在此示例中之后此代码将会被其他函数调用。

```
uint16_t AccumulateCRC(uint8_t data, uint16_t crc)
{
  crc = (uint8_t)((inputCrc >> 8) | (inputCrc << 8));
  crc ^= data);
  crc ^= (uint8_t)(crc & 0xFF) >> 4;
  crc ^= (crc << 8) << 4;
  crc ^= ((crc & 0xFF) << 4) << 1;
  return crc;
}

void AppendCrc(uint8_t[] buffer)
{
  uint16_t crcWord = 0xFFFF;
  int i;
  crcWord = AccumulateCRC(addr, crcWord);
  crcWord = AccumulateCRC(num, crcWord);
  crcWord = AccumulateCRC(cmd[0], crcWord);
  crcWord = AccumulateCRC(dev, crcWord);
  for (i = 0; i < buffer[1]-2; i++) // 通过缓冲区循环减去 CRC
  {
      crcWord = AccumulateCRC(buffer[i], crcWord);
  }
  buffer[i] = (crcWord >> 8) & 0xFF;   // CRC 高字节
  buffer[++i] = crcWord & 0xFF;              // CRC 低字节
}
```

创建消息

下面的代码将创建一条消息，发送到默认地址9的线路板。消息在tx_buffer中被创建，然后被传递到CRC代码以计算并附加CRC字节。

该简单示例直接创建命令包。在实际的程序中，您将通过传递cmd和任何需要的参数来创建一个更通用的解决方案，并将它们放在缓冲区中。

```
uint8_t addr = 9;            // 默认地址
uint8_t tx_buffer[18];       // 足够容纳最长的数据包
uint8_t output_buf[38];      // 足够容纳最长的 ASCII 编码数据包

 void ComposeMessage()
 {
   uint8_t cmd = 0x7E;        // cmd 设置流速
   uint8_t dev = 0;           // 设备始终为 0
   uint32_t flow = 5000000;   // 流速 5 mL / min（单位为 nL / min）
   uint8_t length = 5 + 4;    // 最小长度为 5 (len + cmd + dev
                              // + 2 CRC), + 4 字节参数
   int i;
   // 编写基本消息，将 flow 参数分成 4 个字节
   uint8_t msg[8] = {addr,length,cmd,dev,(flow>>24)&0xFF, (flow>>16)&0xFF,
                     (flow>>8)&0xFF, flow&0xFF};
   for (i=0; i<8; i++)        // 将信息字节传输到 tx_buffer
     tx_buffer[i] = msg[i];
   AppendCrc(tx_buffer);
 }
```

下面是一个创建I²C消息的示例。在调用上面的代码来组成基本的二进制消息之后，接下来要做的就是调整地址以形成一个真正的I²C写入地址，然后将整个数据包复制到输出缓冲区，准备发送。

```
void I2CMessage()
{
    int i;
    ComposeMessage();              //创建添加了 CRC 的基本消息
    output_buf[0] = addr << 1;     // 将地址左移 1 以创建 I2C 写入地址
    for (i = 1; i < tx_buffer[1]; i++)
    {
        output_buf[i] = tx_buffer[i];    // 将其余消息复制到输出中          }
    //这里您将调用代码以将 output_buf 传输到 I2C 端口}
```

创建UART数据包

UART稍微复杂一些，因为它需要对地址后面的所有数据包进行ASCII编码，并在末尾添加回车符。在下面的代码中，请注意它是如何开始使用相同的ComposeMessage代码，然后调用该代码将二进制字节转换为十六进制编码的字节。

```
uint8_t ConvertToHexadecimal(uint8_t c)
{
  if (c < 10)
      c += 0x30;   // 转换为 ASCII 字符“ 0”至“ 9”
else
      c += 0x37;   // 转换为 ASCII 字符“ A”至“ F”
return c;
}

uint16_t bin2hex(uint8_t data)
{
  uint16_t hex;
  uint8_t c;
  hex = ConvertToHexadecimal(data >> 4) << 8;     // 转换高 4 位
  hex |= ConvertToHexadecimal(data & 0xF);        // 转换低 4 位
return hex;
}

void ComposeMessage()
{
  uint8_t cmd = 0x7E;           //  cmd设置流速
  uint8_t dev = 0;              // 设备始终为0
  uint32_t flow = 5000000;      // 流速5 mL / min（单位为nL / min）
  uint8_t length = 5 + 4;       // 最小长度为5（len + cmd + dev
                                // + 2 CRC），+ 4字节参数
  int i;
  // 编写基本消息，将flow参数分成4个字节
  uint8_t msg[8] = {addr,length,cmd,dev,(flow>>24)&0xFF,
(flow>>16)&0xFF,
                    (flow>>8)&0xFF, flow&0xFF};
  for (i=0; i<8; i++)           // 将信息字节传输到tx_buffer
    tx_buffer[i] = msg[i];
  AppendCrc(tx_buffer);
}
```

读取电路板的响应

上面的例子没有说明如何从电路板上读取消息。对于I²C，您将发送读取地址 [基址左移1位，然后设置位0，((addr<<1) | 1)]。然后继续读取字节（对SCL行时钟响应），直到收到响应中预期的字节数。

当读取UART响应时，您需要将ASCII编码的数据包字节转换回二进制，然后解释响应的含义。下面是一个代码示例，它将两个十六进制字符转换为等效的二进制值。

```
uint8_t hex2bin(uint8_t c1, uint8_t c2)
{
    uint8_t b;              //这是作为结果的二进制字节
    if (c1 < 'A')           // 如果第一个字符为“0”至“9”
        b = c1 - '0';       // 转换为二进制值 0 到 9
    else                    // 否则是“A”到“F”
        b = c1 - 'A' + 10;    // 转换为二进制值 10 到 15
    b = b << 4;               // 将结果放在高 4 位

    if (c2 < 'A')           // 对第二个字符进行相同的操作
        b += c1 - '0';      // 并将其添加到低 4 位
    else
        b += c1 - 'A' + 10;

    return b;
}
```