AN-2033
应用笔记

One Technology Way • P.O. Box 9106 • Norwood, MA 02062-9106, U.S.A. • Tel: 781.329.4700 • Fax: 781.461.3113 • www.analog.com/cn

校准ADPD188BI光学烟雾和气雾剂检测模块

简介

ADPD188BI是完整的光电式测量系统，适合采用光学双波长技术的烟雾检测应用。该模块集成了高效率光电式测量前端、蓝光和红外(IR)发光二极管(LED)以及一个光电二极管，这些元件包含在定制封装中，光线必须先进入烟雾检测室，而无法直接从LED到达光电二极管。ADPD188BI与EVAL-CHAMBER烟雾室配合使用，可创建一套完整的光学烟雾检测解决方案，适用于住宅和工业烟雾探测器。EVAL-CHAMBER可通过订购EVAL-ADPD188BIZ-S2购买。

本应用笔记说明如何使用已写入片内非易失性存储器(NVM)的校准系数来校准ADPD188BI，从而将器件之间的差异降低至±10%以下。

对于特定的LED驱动器设置和测试/应用环境，ADPD188BI的LED响应会表现出器件间差异。LED响应的斜率（增益）和截距（失调）随器件而异，导致不同器件对同一环境的响应存在差异，这可以利用增益和失调校准系数进行校准。这种校准的主要应用是可以对最终应用中实例化的多个器件的输出进行更有效的比较。此校准可显著减少器件之间的光学变化带来的测量差异，并简化对应用环境特定的变化的观测。

目录

修订历史

2021年2月—修订版0至修订版A

更改“简介”部分 1

添加了“校准 ADPD188BI”部分 3

更改“读取 eFuse 寄存器”部分和表 1 3

将计算校准系数更改为计算模块 ID 30 和模块 ID 31 的校准系数 3

添加了“计算模块 ID 33 的校准系数”部分、“校准 32 kHz 和 32 MHz 振荡器以获得出色的系统性能”部分、“基于模块 ID 应用正确的方程”部分和表 2；重新排序 4

更改图 1 和图 2 5

更改“ECC 的 C 代码”部分 8

2019年12月—修订版0：初始版

校准ADPD188BI

测试方法

每对LED/驱动器会有多个LED电流进入反射器，反射器的响应由ADPD188BI模块内部的光电二极管测量。针对每对LED/驱动器计算响应的斜率，截距从线性回归得出。然后计算校准系数，并将其存储在片内NVM（也称为eFuse寄存器）中，以供以后在最终应用中使用。校准系数基于特定器件的每脉冲测量值计算，归一化为从不同器件收集到的大量数据分布的平均值。这种归一化确保了大量器件中器件间的差异最小。

读取eFuse寄存器

失调和增益校准系数存储在片内eFuse寄存器中。增益校准系数LED1_GAIN_COEFF和LED3_GAIN_COEFF分别存储在寄存器0x71和0x72中。失调校准系数LED1_INT_COEFF和LED3_INT_COEFF分别存储在寄存器0x73和寄存器0x74中。

要访问eFuse寄存器，请执行以下步骤：

1. 设置寄存器0x4B的位7 = 1以使能32 kHz振荡器。
2. 将0x1写入寄存器0x10以强制器件进入编程(空闲)模式。
3. 将0x1写入寄存器0x5F以使能32 MHz先进先出(FIFO)时钟。
4. 将0x7写入寄存器0x57以使能对eFuse寄存器的访问。
5. 读取寄存器0x67。当寄存器0x67 = 0x04时，eFuse寄存器的刷新完成，可以随时对其进行读取。
6. 应用校准系数之前，须对eFuse数据应用纠错码(ECC)函数（参阅“使用ECC检测和校正EFUSE值的错误”部分）。
7. 确认模块ID 30、模块ID 31、模块ID 33或以上的寄存器0x70的内容分别为0x1E、0x1F、0x21或更大。
8. 读取所需LED/驱动器对的增益和失调校准系数。使用eFuse寄存器的内容，按照“计算校准系数”部分中所述计算最终增益校准系数。得出最终增益校准系数后，将其载入用户可访问的存储器中以备将来使用。
9. 完成对eFuse寄存器的读取后，按照以下方式禁用eFuse寄存器：
   a. 将0x0写入寄存器0x57以禁用对eFuse寄存器的访问。
   b. 将0x0写入寄存器0x5F以禁用32 MHz FIFO时钟。

计算模块ID 30和模块ID 31的校准系数

最终校准系数必须使用寄存器0x71至寄存器0x74的内容来计算，如下式所示：

$$GAIN\_CAL\_X = DEVICE\_SCALAR/NOMINAL\_SCALAR$$

其中：

DEVICE_SCALAR = *x_GAIN* × *LEDx* + *x_INTERCEPT*。

*x_GAIN*对于蓝光LED通道为BLUE_GAIN，对于IR LED通道为IR_GAIN。

BLUE_GAIN = (17/256)(LED1_GAIN_COEFF − 112) + 17。

IR_GAIN = (34/256)(LED3_GAIN_COEFF − 112) + 34。

*LEDx*是以毫安为单位的LED驱动电流；例如，若驱动电流 = 200 mA，则输入200。*LEDx*对于蓝光LED通道为LED1，对于IR LED通道为LED3。

*x_INTERCEPT*对于蓝光LED通道为BLUE_INTERCEPT，对于IR LED通道为IR_INTERCEPT。

BLUE_INTERCEPT = 8(LED1_INT_COEFF − 128)。

IR_INTERCEPT = 5(LED3_INT_COEFF − 128)。

NOMINAL_SCALAR = x_MEAN_GAIN × LEDx + x_MEAN_INTERCEPT。

x_MEAN_GAIN对于蓝光LED通道为17，对于IR LED通道为34。

x_MEAN_INTERCEPT对于蓝光LED通道为622，对于IR LED通道为128。

表1. 模块ID 30和ID 31的eFuse寄存器的内容

地址	名称	位	描述
0x70	MODULE_ID	[7:0]	模块ID = 30或31
0x71	LED1_GAIN_COEFF	[7:0]	蓝光LED增益系数
0x72	LED3_GAIN_COEFF	[7:0]	IR LED增益系数
0x73	LED1_INT_COEFF	[7:0]	蓝光LED截距系数
0x74	LED3_INT_COEFF	[7:0]	IR LED截距系数
0x7E	ECC	[7:0]	ECC

计算模块ID 33的校准系数

最终校准系数必须使用寄存器0x71至寄存器0x74的内容来计算，如下式所示：

GAIN_CAL_X = *DEVICE_SCALAR*/*NOMINAL_SCALAR*

其中：

DEVICE_SCALAR = *x_GAIN* × *LEDx* + *x_INTERCEPT*。

*x_GAIN*对于蓝光LED通道为BLUE_GAIN，对于IR LED通道为IR_GAIN。

BLUE_GAIN = (21/256)(LED1_GAIN_COEFF − 112) + 21。

IR_GAIN = (42/256)(LED3_GAIN_COEFF − 112) + 42。

*LEDx*是以毫安为单位的LED驱动电流；例如，若驱动电流 = 200 mA，则输入200。*LEDx*对于蓝光LED通道为LED1，对于IR LED通道为LED3。

*x_INTERCEPT*对于蓝光LED通道为BLUE_INTERCEPT，对于IR LED通道为IR_INTERCEPT。

BLUE_INTERCEPT = 8(LED1_INT_COEFF − 80)。

IR_INTERCEPT = 5(LED3_INT_COEFF − 80)。

NOMINAL_SCALAR = x_MEAN_GAIN × LEDx + x_MEAN_INTERCEPT。

x_MEAN_GAIN对于蓝光LED通道为21，对于IR LED通道为42。

x_MEAN_INTERCEPT对于蓝光LED通道为753，对于IR LED通道为156。

校准32 KHz和32 MHz振荡器以获得出色的系统性能

校准32 KHz和32 MHz片内振荡器以获得出色的性能。32 kHz振荡器确定ADPD188BI的整体采样速率，32 MHz振荡器则影响ADPD188BI的总增益。对于模块ID = 33的器件，读取eFuse寄存器（寄存器0x77和寄存器0x78），并将这些值分别写入器件寄存器0x4B和寄存器0x4D。或者，用户也可按照ADPD188BI数据手册中所述的32 kHz和32 MHz时钟校准程序手动确定最优设置。

基于模块ID应用正确的方程

为实现最佳操作，读取eFuse寄存器0x70以确定模块ID并应用合适的方程。下面是用户软件一部分的条件语句示例。

```
// check module ID
Case (Module ID):
       // For IDs 30 & 31
       Case 30, 31:
              GAIN_CAL_BLUE = (use equations
shown in Calculating Calibration Coefficients
for Module ID 30 & Module ID 31)
              GAIN_CAL_IR = (use equations
shown in Calculating Calibration Coefficients
for Module ID 30 & Module ID 31)
       // For ID 33
       Case 33:
              GAIN_CAL_BLUE = (use equations
shown in Calculating Calibration
Coefficients for Module ID 33)
              GAIN_CAL_IR = (use equations
shown in Calculating Calibration
Coefficients for Module ID 33)
       Case TBD1: leave for future expansion
       Case TBD2: leave for future expansion
       Default: raise error
```

表2. 模块ID 33的eFuse寄存器的内容

地址	名称	位	描述
0x70	MODULE_ID	[7:0]	模块ID = 33
0x71	LED1_GAIN_COEFF	[7:0]	蓝光LED增益系数
0x72	LED3_GAIN_COEFF	[7:0]	IR LED增益系数
0x73	LED1_INT_COEFF	[7:0]	蓝光LED截距系数
0x74	LED3_INT_COEFF	[7:0]	IR LED截距系数
0x77	32kHz_OSC_OPT_ADJUST	[7:0]	32 kHz振荡器最优调整设置
0x78	32MHz_OSC_OPT_ADJUST	[7:0]	32 MHz振荡器最优调整设置
0x7E	ECC	[7:0]	ECC

应用校准系数

要在最终应用中应用校准系数，请执行以下步骤：

1. 根据需要配置ADPD188BI器件。
2. 将0x2写入地址0x10以开始正常采样操作。
3. 在所需的LED电平下进行测量并执行以下计算：

   归一化输出(LSB) = AFE_OUT/GAIN_CAL_x

其中：

AFE_OUT = LED亮起时的原始输出测量值。

*GAIN_CAL_x*对于蓝光 LED 通道为GAIN_CAL_BLUE，对于IR LED通道为GAIN_CAL_IR。

应用校准系数可大大减小器件之间的差异。图1和图2显示了校准前后蓝光LED和IR LED的直方图。图1和图2说明，在两种情况下，器件间差异的分布范围缩小到±10%以下。

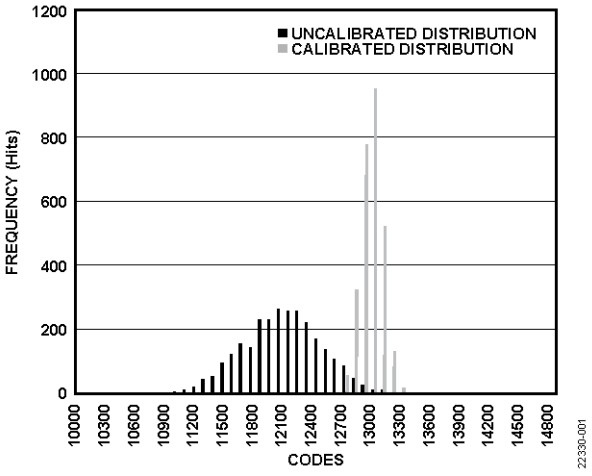

图1. 校准前后的蓝光LED响应

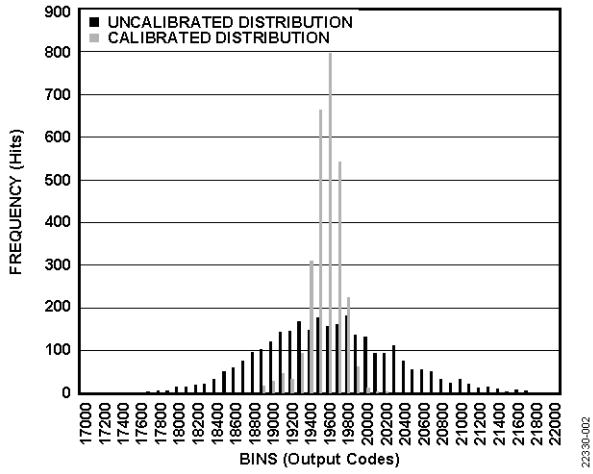

图2. 校准前后的IR LED响应

eFuse内容对器件正常操作的影响

写入ADPD188BI eFuse寄存器的校准系数不会改变器件性能或规格。所有数据手册规格和器件性能本质上不受eFuse寄存器编程的影响。

校准系数旨在用于采样数据的后处理，以便校准器件之间的光学特性差异。无论对eFuse寄存器进行编程与否，ADPD188BI的性能都没有差异。在eFuse寄存器写有校准系数的情况下，只有在软件中对采样数据实施后处理校准程序时，eFuse寄存器中存储的数据才会对最终数据产生影响。

使用ECC检测并校正eFuse值的错误

“ECC的C代码”部分中显示的C代码包含了使用汉明码检测和校正存储的eFuse寄存器值中错误的例程。这些函数使用传统的127120海明码，截断为119112。其中添加了一个额外的全局奇偶校验位，以实现2位故障检测以及1位校正。最终形式是120,112，它将8位奇偶校验码添加到每个112位(14字节)块中。

此代码可100%检测并修复每个数据块中的1位错误，并能100%检测到每个数据块中的2位故障。

方法如下：将eFuse数据和奇偶校验字节读入本地存储器。用户必须读取寄存器0x70至寄存器0x7E。寄存器0x70至寄存器0x7D与输入指针和数据相关联，必须将其读入数据数组。寄存器0x7E与输入指针和奇偶校验相关联，必须作为奇偶校验值读入。使用fix_hamm_parity命令验证该块。此函数会在原位修复单个损坏的位。如果fix_hamm_parity命令返回错误，则将器件标记为已损坏。

此过程可修复所有1位故障，检测到所有2位故障和约6%的3位故障，并检测到大多数偶数故障。

回流焊对校准系数的影响

在氧含量不受控制的回流炉中进行回流焊时，可能导致光电二极管对蓝光LED的响应降低。平均而言，每次回流，光电二极管对蓝光LED的响应会有约7%的偏移。校准系数是在最终测试时写入的，先于ADPD188BI的任何回流。因此，如果ADPD188BI在含氧量不受控制的炉中进行回流焊，蓝光系数便不再准确。

图3显示了在氧含量不受控制的烤箱中回流后的原始蓝光响应和校准后的响应。这组器件经过了三次回流。数据包括每次回流后的测试数据。从数据中可以看出，每次回流后，蓝光LED响应约有7%的偏移。

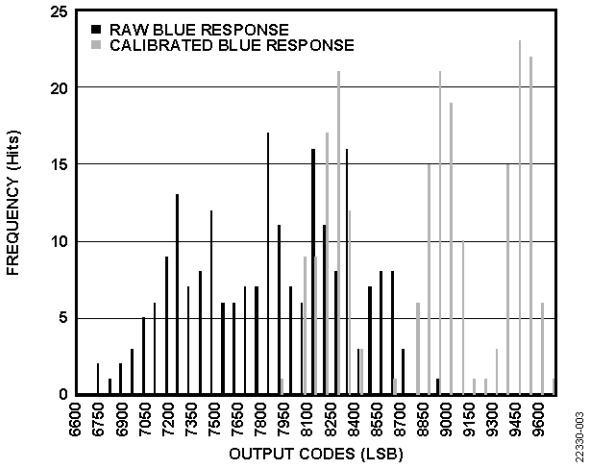

图3. 回流焊期间，不受控制的氧含量引起的蓝光LED响应的偏移

为避免响应偏移，应使用通过氮气降低炉中氧含量的回流炉。当使用氮气控制的回流炉将氧含量控制在1000 ppm以下时，蓝光LED响应不会因为回流焊发生偏移。

图4中的数据显示了已在炉中回流三次的器件的原始蓝光LED响应值和校准后的响应值，该炉使用氮气吹扫，氧含量降低至1000 ppm以下。数据包括每次回流后的测试数据。如图4所示，在这些情况下，响应没有因回流而发生偏移。

无论炉中的氧含量是否受到控制，IR响应都不受回流影响。

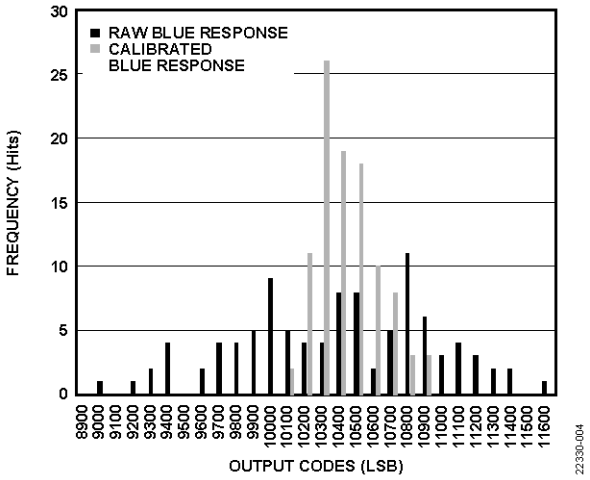

图4. 回流焊期间，通过氮气吹扫控制氧含量的情况下蓝光LED响应的偏移

ECC的C代码

```
int generate_hamm_block_parity( data )
     int data[];
{
// Define parity mapping for parity byte generation/testing
// traditional hamming coding for 127,120 truncated to 120,112
// plus extra parity to make 120,112 code for SECDED.
//
// this table determines which parity bits are involved in each data bit.
// MSB is global "all data parity"
// this function does not include the parity bits in the global bit
// so it can be added differently in the generate_hamm_parity
// and generate_hamm_syndrome functions as needed

  const int paritymap[112]={
    131,  133,  134,  135,  137,  138,  139,  140,  141,  142,
    143,  145,  146,  147,  148,  149,  150,  151,  152,  153,
    154,  155,  156,  157,  158,  159,  161,  162,  163,  164,
    165,  166,  167,  168,  169,  170,  171,  172,  173,  174,
    175,  176,  177,  178,  179,  180,  181,  182,  183,  184,
    185,  186,  187,  188,  189,  190,  191,  193,  194,  195,
    196,  197,  198,  199,  200,  201,  202,  203,  204,  205,
    206,  207,  208,  209,  210,  211,  212,  213,  214,  215,
    216,  217,  218,  219,  220,  221,  222,  223,  224,  225,
    226,  227,  228,  229,  230,  231,  232,  233,  234,  235,
    236,  237,  238,  239,  240,  241,  242,  243,  244,  245,
    246,  247 };
  //
  int bit,byte; // pointers
  int h;        // parity byte

  h=0; // init parity byte
  // calculate parity for the 112 data bits according to map
  for(byte=0; byte < 14; byte++) {
    for(bit=0x0; bit < 8 ; bit++) {
      if( (data[byte] & (1<<bit) )!=0){ h ^= paritymap[(byte<<3)+bit];
      }
    }
  }
  return(h); // return the parity byte for the 112bit block only
}
//
//
int generate_hamm_syndrome( data, parity_in )
     int data[],*parity_in;
{
  //
```

```
  // generate final hamm parity using two steps
  // - generate parity for 112 bit data block
  // - include input parity into global parity bit
  //
  int bit; // pointer
  int h;   // parity byte

  h=generate_hamm_block_parity(data);  // get parity byte for 112 bits

  // add the parity of the 7 input parity bits into the global
  for(bit=0;bit<6;bit++) {
    if ((*parity_in&(1<<bit))==(1<<bit)) h^=0x80;
  }
  return(h); // return the final parity
}
//
// This function checks the data and parity byte
// for consistency and corrects single bit problems
// Return Values:
//   - 0 if the data/parity is correct. (NO REPAIR DONE)
//   - 1 if there is a single bit error in the data region (REPAIRED)
//   - 2 if there is a single bit error in the parity byte (REPAIRED)
//   - 3 if there are multiple errors (NO REPAIR DONE)
//
int fix_hamm_parity (data, parity)
     int data[]; int *parity;
{
  int calculated_parity;
  int syn, glob;
  int bit, byte;

  calculated_parity=generate_hamm_syndrome(data,parity);
  syn=(*parity^calculated_parity)&0x7f;
  glob=(*parity^calculated_parity)&0x80;
  if(glob==0) {
    if (syn==0) return(0); // no errors (no fix needed)
    else return(3); // double error (can't fix)
  }
  else {
    if (syn>=120) return(3); //also double error
    switch (syn) { // error in lower parity (fix the bit)
    case 0:  *parity=*parity ^ 0x80; return(2);
    case 1:  *parity=*parity ^ 0x01; return(2);
    case 2:  *parity=*parity ^ 0x02; return(2);
    case 4:  *parity=*parity ^ 0x04; return(2);
    case 8:  *parity=*parity ^ 0x08; return(2);
    case 16: *parity=*parity ^ 0x10; return(2);
```

```
    case 32: *parity=*parity ^ 0x20; return(2);
    case 64: *parity=*parity ^ 0x40; return(2);
    default:       // error in data block (fix it)
// if it gets here there is a single bit data error
// first adjust the address to account for the
// parity bits being outside the data region
      syn =
        (syn>64) ? syn - 8 :
        (syn>32) ? syn - 7 :
        (syn>16) ? syn - 6 :
        (syn>8)  ? syn - 5 :
        (syn>4)  ? syn - 4 : 0;

      byte = syn >> 3;
      bit = syn & 0x7;
      data[byte]=data[byte]^(1<<bit);  // fix the data bit
      return(1);         // single data error (fixed)
    }
  }
}
```